DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE FICHEROS SIMPLE CON FUSE

Provecto de fin de carrera

Autor Carlos Martínez Luque
Tutor Javier Fernández Muñoz

Resumen

Este proyecto describe el desarrollo de una aplicación que permita montar varios sistemas de almacenamiento en un mismo punto de montaje. Se diseñarán diferentes módulos y estructuras de datos para la gestión de las diferentes operaciones soportadas. El presente documento tiene como finalidad la presentación del proceso completo llevado a cabo para realización del proyecto.

Abstract

The Project carried out describes the development of an application able to mount various storage devices in the same mount point. Several modules and data structures will be designed to manage all supported operations. The current document details the complete process needed to create the project.

Índice

Índice de Ilustraciones

Índice de Tablas

1 Introducción

La idea en la que se basa el proyecto consiste en poder utilizar, de forma transparente para el usuario, un conjunto de unidades de almacenamiento montadas en un equipo como si fueran una sola.

En este punto es en el que entra en juego *FUSE*, un conjunto de librerías que permite crear sistemas de ficheros en espacio de usuario. Esta herramienta va a permitir crear un sistema de archivos virtual en el que se permitirá el acceso, de forma unificada, de las unidades que se monten en él.

Este proyecto abarca la primera fase de desarrollo, en la cual se implementarán una serie de funciones básicas para la creación y borrado de archivos. Además, se creará un módulo que se encargue de gestionar la política de elección de sistemas de ficheros. Para esta fase, dicha elección se realizará utilizando *Round Robin*. Es importante que este módulo tenga un nivel de acoplamiento bajo con el resto del sistema ya que, en siguientes versiones del sistema, se implementarán nuevas políticas.

El desarrollo será realizado utilizando *SCRUM* y abarca las seis primeras iteraciones sobre el producto. Durante este periodo, se mantendrán una serie de reuniones periódicas con el cliente, en las que se validará el progreso realizado.

1.1 Objetivos

El objetivo principal de este proyecto consiste en crear la base para una aplicación que permita al usuario interactuar con un grupo de unidades de almacenamiento montadas previamente, como si se tratase de un único sistema de ficheros.

Este objetivo general, engloba una serie de objetivos secundarios enumerados a continuación, los cuales permitirán el desarrollo de la primera versión del sistema y la consecución del objetivo principal. Son los siguientes:

- Desarrollo de un núcleo de operaciones con directorios que abarca la creación, borrado y acceso al contenido de los mismos.
- Desarrollo de un núcleo de operaciones con archivos. El conjunto de funcionalidades debe incluir la creación, borrado, copiado al sistema de ficheros creado, renombrado de ficheros y acceso a la información que almacenan los archivos.
- Implementación de un módulo que implemente una política de asignación de sistemas de ficheros básica.

Para la consecución de las metas propuestas, es necesario que se consiga alcanzar los siguientes sub-objetivos:

- Para facilitar la implementación de nuevas políticas de asignación de sistemas de ficheros, el módulo encargado debe tener un nivel bajo de acoplamiento con el resto del código de la aplicación.
- El sistema debe llevar una trazabilidad sobre las acciones ocurridas sobre el sistema de ficheros creado.
- El sistema debe informar sobre las operaciones no permitidas y los errores ocurridos en la ejecución de las acciones.

1.2 Estructura del documento

A continuación, se describen brevemente el contenido de los capítulos del documento

Capítulo 1.

Aporta una visión global del proyecto y describe los objetivos a cumplir en su desarrollo.

Capítulo 2

En este capítulo se describen los conceptos, tecnologías y metodologías de trabajo necesarias para la comprensión y desarrollo del proyecto.

Capítulo 3

Detalla el resultado de la fase de análisis de las necesidades del sistema. Se enumerarán y detallarán los requisitos obtenidos y los casos de uso que definirán el comportamiento de la aplicación.

Capítulo 4

Se describen las decisiones técnicas, estructuras de datos y detalles técnicos que guiarán la fase de desarrollo.

Capítulo 5

Se describen las tareas a realizar, criterios de aceptación y sus estimaciones. Además, incluye un plan de despliegue que describe el orden en el que serán liberadas cada una de las versiones.

Capítulo 6

Muestra el resultado de las reuniones de demostración en las que se enumeran las funcionalidades incluidas en la versión analizada y los resultados de sus test en función de los criterios de aceptación.

Capítulo 7

Lista la estimación de gastos en personal, software y hardware que serán incluidos en el presupuesto.

Capítulo 8

Se evalúa la consecución de los objetivos marcados en el capítulo 1 y se proponen futuras líneas que podría seguir el desarrollo de la aplicación.

2 Estado de la cuestión

2.1 Espacio de usuario y espacio del *kernel*

Existe una separación entre los recursos utilizados por el núcleo del sistema y el resto. De esta forma, el código y recursos utilizados por el *kernel* definen lo que comúnmente se denomina *espacio del kernel*. En consecuencia, todo lo que sucede fuera del *espacio del kernel* es denominado *espacio de usuario*.

Normalmente, los procesos que se ejecutan en *espacio del kernel*, sólo pueden ser lanzados por usuarios con privilegios ya que, estos procesos tienen acceso a todos los componentes del sistema y todas las instrucciones de la máquina.

Por el contrario, los procesos lanzados en el *espacio de usuario*, tienen acceso a una parte limitada de la memoria. Además, estos no tienen acceso al *espacio del kernel* salvo, a través de una serie de funciones expuestas por el sistema. Estas son las llamadas *system calls* o *llamadas de sistema*.

Cuando un proceso realiza una *llamada de sistema*, su ejecución se interrumpe y sus datos son guardados. En ese momento, se empiezan a ejecutar las acciones requeridas, con los privilegios necesarios, en el espacio del *kernel*. Al concluir, el proceso recupera sus datos y continúa con su ejecución en el *espacio de usuario*. Mediante este mecanismo, un proceso en *espacio de usuario* puede acceder a recursos que sólo son accesibles en el *espacio del kernel*.

Cómo introducción a lo que se expondrá en las siguientes secciones, es conveniente mencionar que, en los procesos relacionados con el uso de sistemas de ficheros, una acción, por ejemplo, la escritura en un archivo, puede comenzar en el *espacio de usuario* pero, las librerías necesarias para realizarla, se ejecutarán en el *espacio del kernel* mediante el uso de *llamadas de sistema*.

2.2 Sistemas de ficheros

Actualmente, existe un gran número de soportes de almacenamiento de datos (discos magnéticos, discos ópticos, tarjetas de memoria, etc…). Cada uno de ellos utiliza un soporte de almacenamiento diferente y sus propias estructuras físicas para almacenar los datos. Por ejemplo, en el caso de los discos duros, los datos son almacenados en una cinta magnética como pulsos que serán interpretados como ceros y unos. En el caso de los soportes ópticos, estos datos son leídos a través de las propiedades *óptico-reflectivas* de una sección del disco. Estos datos necesitan una estructura para poder discernir el punto en el que comienzan y terminan cada una de las piezas de información.

Para dar coherencia al conjunto de datos en bruto, es necesario agruparlos en piezas más pequeñas y que tengan sentido por sí mismas. A cada uno de estos grupos se les llamará archivos. Las reglas usadas para delimitar estos grupos y su estructura será llamada sistema de ficheros. En el siguiente esquema, se muestra la estructuración que realiza *EXT2*.

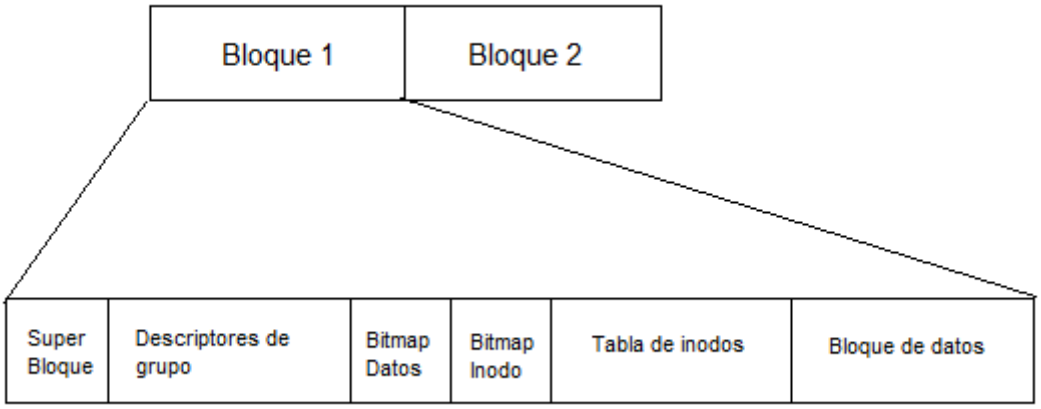


Ilustración 1 Estructura EXT2

Un sistema de ficheros es un conjunto de métodos y estructuras de datos que un sistema operativo utiliza para almacenar y organizar datos, haciendo más simple las tareas de búsqueda y acceso. Principalmente, sus funciones abarcan la administración y asignación de espacio a los archivos alojados en él, administración de espacio libre y acceso a los datos almacenados.

Hay una gran cantidad de sistemas de ficheros y, cada uno de ellos, puede ser utilizado en diferentes sistemas de almacenamiento. Cada sistema de ficheros define sus propias reglas en cuanto a la estructura y lógica de los archivos almacenados. En algunos casos, como ocurre con el estándar *ISO 9660*, estos sistemas de ficheros serán diseñados para ser utilizados en un tipo de soporte concreto (soporte óptico en el caso del *ISO 9660*).

2.2.1 Archivos y directorios

Para el sistema de ficheros, un archivo es una colección de bytes con un formato concreto. Cada archivo estará localizado de forma única, mediante un nombre de archivo o *filename*. La estructura del *filename*, tanto sus restricciones en cuanto a tamaño cómo los caracteres de los que puede constar, vendrán determinadas por el sistema de ficheros. En función del sistema operativo, el *filename* incluirá estos componentes:

- **Host**: Dispositivo de red que contiene el archivo.
- **Dispositivo**: Dispositivo de hardware en el que se encuentra.

- **Directorio**: Árbol de directorios.
- **Archivo**: Nombre del archivo.
- **Tipo**: Indica el tipo del contenido
- **Versión**: revisión del archivo.

Un directorio, permite agrupar archivos en colecciones. Su estructura puede ser plana o, en el caso de tratarse de un sistema de ficheros jerárquicos, contener subdirectorios. Es habitual referirse a la relación entre un directorio y los directorios que contiene, como padre e hijos. En esta jerarquía, el directorio que se encuentra en el nivel más alto de la estructura, será llamado raíz o *root*.

2.3 Sistema de ficheros virtual en LINUX.

El sistema de ficheros virtual, o *VFS*, es un subsistema dentro del *kernel* de los sistemas *UNIX* que aporta una serie de interfaces para abstraer y simplificar la interacción entre las aplicaciones utilizadas por el usuario y el sistema de ficheros. De esta forma, el *VFS* permite trabajar, utilizando llamadas al sistema, con independencia del utilizado o los medios físicos que intervengan.

El *VFS* se sitúa como una capa intermedia entre el espacio de usuario y los diferentes sistemas de ficheros consiguiendo que, llamadas las sistema como *open(), read()* y *write()*, funcionen a través de ellos. Este desacoplamiento de la implementación a bajo nivel de un sistema de ficheros del resto del *kernel* permite simplificar la tarea de integrar un nuevo sistema de ficheros dentro del sistema se reduce a implementar el contrato, la interfaz, proporcionada por el sistema de ficheros virtual.

El modo de funcionamiento habitual del *VFS* sigue los siguientes pasos:

1. Se produce una llamada del sistema que implica un acceso al sistema de ficheros.
2. El *VFS* determina a cuál de los sistemas de ficheros montados corresponde la llamada.
3. Se comprueba el estado de la cache en busca de una entrada existente y válida.
4. Si es necesario, en este paso el *VFS* se ejecutará la función del sistema de ficheros específico que realiza la función solicitada. La ejecución de esta función puede completar la llamada, o bien requerir una operación de entrada/salida sobre el dispositivo correspondiente.

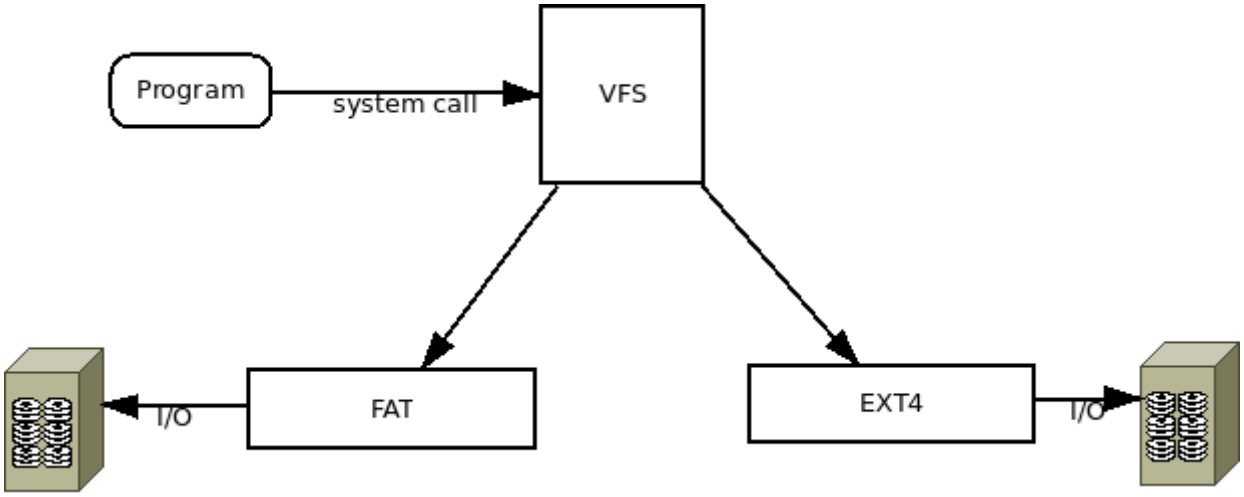


Ilustración 2 Estructura VFS

Esta capa de abstracción funciona definiendo una serie conceptos en forma de interfaces y estructuras de datos que todos los sistemas de ficheros soportan. Como resultado, se obtiene una capa de abstracción que permite al *kernel* soportar varios sistemas de ficheros sin necesidad de conocer los detalles sobre los mismos. Además, permite al espacio del usuario interactuar con ellos de forma clara y sencilla.

Los conceptos estructurales relativos, como archivos y directorios, tienen estructuras físicas y lógicas en función del sistema de ficheros implementado. Para abstraer estos conceptos, VFS provee de un modelo común para archivos (*common file model* o *CFM*). Esta familia de estructuras contienen datos y punteros a las funciones implementadas por el sistema de ficheros y que operan sobre los datos.

Las cuatro estructuras principales son:

- **Superblock**, que representa a un sistema de ficheros específico, que se encuentra montado.
- **Inode**, que representa a un archivo.
- **Dentry**, que representa a un componente de una ruta.
- **File**, que representa un archivo abierto y asociado a un proceso.

En las siguientes secciones, se describirán estas estructuras con mayor detalle.

2.3.1 Superbloque (Superblock)

El *superbloque* es, esencialmente, el conjunto de los metadatos del sistema de ficheros. Define el tipo del sistema, su tamaño, estado, bloques ocupados y su conteo. Este tipo de estructura es crítica y suele estar almacenada de forma redundante a lo largo del sistema de ficheros.

La estructura *superblock* ofrece, en su interfaz, entre otras, las siguientes definiciones:

- **alloc_inode**: Inicializa y crea un Nuevo *inode* en el *superblock* pasado por parámetro.
- **dirty_inode**: Esta función es llamada por el Sistema virtual de ficheros cuando un inode es modificado.
- **read_inode**: Su función es localizar el bloque del disco que contiene el *inodo* y leerlo.
- **write_inode**: Persiste el *inode* de forma sincrona o asincrona en función de uno de los parámetros.
- **delete_inode**: Borra el *inode* del sistema de ficheros.
- **clear_inode**: Libera los datos relativos a la estructura de un *inodo*.
- **write_super**: Esta función se utiliza para sincronizar el contenido modificado de un *superbloque* con el disco.
- **sync_fs**: Sincroniza los metadatos del *superbloque* con los datos del disco.
- **write_super_lockfs**: Bloquea los cambios en el Sistema de ficheros y actualice el *superbloque* en el disco.
- **unlockfs**: Desbloquea el bloqueo de cambios en el sistema de ficheros.
- **statfs**: Obtiene las estadisticas del sistema de ficheros.
- **remount_fs**: Esta función es llamada por el Sistema cuando el Sistema de ficheros es remontado con nuevos parámetros.
- **put_super**: Esta función es llamada para desmontar el *superbloque* pasado por parámetro.
- **umount_begin**: Está función se encarga de interrumpir el proceso de montado de un Sistema de ficheros

2.3.2 Inodo (Inode)

Un *inodo* es una estructura de datos que almacena toda la información relativa a un archivo excepto su nombre y sus datos. Estos metadatos suelen incluir datos referentes al propietario, métodos de acceso y tipo de archivo.

En sistemas Linux, un *inodo* representa tanto a un directorio, almacenando tablas con los nombres de los archivos que contiene, como archivos, almacenando sus metadatos. Las estructuras de *inodos* son creadas en memoria en el momento en el que los archivos o directorios son accedidos.

En el VFS, todos los *inodes* se combinan en una lista que puede ser accedida a través de un has formado a partir de la información del propio *inode*. Si un *inodo* se necesita y no está en la cache, se busca en el disco y se cachea.

Estos son las definiciones de funciones que ofrece su interfaz:

- **create**: Esta función es llamada para crear un nuevo *inodo* y asociarlo a un objeto de tipo *dentry*.
- **lookup**: Busca en la tabla de *inodos* de un objeto *dentry* el nombre de archivo especificado.
- **link**: Esta funcionalidad permite crear un enlace de un archivo en la tabla de *inodos* de un objeto de tipo *dentry*.
- **unlink**: Elimina la entrada de un archivo de la tabla de *inodos* de un objeto de tipo dentry.
- **symlink**: Crea un enlace simbólico.
- **mkdir**: Crea un nuevo directorio.
- **rmdir**: Borra un directorio referenciado por un objeto de tipo *dentry*.
- **mknod**: Crea un tipo especial de archivo (device, pipe o socket).
- **rename**: Mueve un archivo situado en la primera de las rutas pasadas por parámetro a la segunda.
- **readlink**: Copia un número solicitado de bytes de una ruta a un *buffer*.
- **follow**_link: Obtiene un *inodo* a través de un link que lo apunta.
- **put_link**: Limpia tras llamar a **follow_link**.
- **truncate**: Modifica el tamaño de un fichero.
- **permission**: Comprueba si un *inodo* es accesible a través de un modo de acceso concreto.
- **setattr**: Notifica cuando un *inodo* ha sido modificado.
- **getattr**: Obtiene los atributos de un *inodo*.
- **setxattr**: Asigna atributos extendidos a un *inodo*.
- **getxattr**: Obtiene los atributos extendidos de un *inodo*.
- **listxattr**: Copia la lista de atributos extendidos a un *buffer*.
- **removexattr**: Elimina un atributo extendido de un *inodo*.

2.3.3 Dentry

Como se describe en la sección anterior, el sistema virtual de ficheros trata a directorios con el mismo tipo que a los archivos. Por ejemplo, dada la ruta */tmp/file.txt* tanto *tmp* como *file.txt* son tratados como archivos y un *inode* representará a cada uno de estos componentes. A pesar de esta unificación, *VFS* necesita realizar operaciones específicas a nivel de directorio.

Para simplificar esta tarea, *VFS* implementa el concepto de *dentry* (*directory entry*). Una estructura de tipo *dentry* representa a un componente concreto dentro de una ruta (archivo o directorio). Cuando se lee una entrada de un directorio, el *kernel* crea un objeto de tipo *dentry* por cada componente de su *inodo*.

A continuación, se listan las funciones de los objetos de este tipo:

- **d_revalidate**: Determina si un objeto de tipo *dentry* es válido.

- **d_hash**: Crea un identificador para un objeto de tipo *dentry*
- **d_compare**: Compara dos *filenames*.
- **d_delete**: Borra un *dentry*.
- **d_release**: Desbloquea a un objeto de tipo *dentry*.
- **d_iput**: Es llamada cuando un objeto de tipo *dentry* pierdo su *inodo* correspondiente.

2.3.4 File

Este tipo de objetos son utilizados para representar a un archivo que está siendo utilizado por un proceso. Esto implica que por cada archivo abierto va a existir un objeto de tipo *file*. Estas estructuras de datos, almacenarán datos específicos sobre la instancia abierta y el usuario.

Las operaciones que permite realizar son las siguientes:

- **llseek**: Actualiza el puntero a un archivo.
- **read**: Lee un número concreto de bytes de un archivo y los vuelca en un *buffer*.
- **aio_read**: Realiza una lectura asíncrona un archivo.
- **write**: Escribe un número dado de bytes leídos de un *buffer* a un archivo.
- **aio_write**: Comienza una escritura asíncrona de *bytes* a un archivo.
- **readdir**: Devuelve el siguiente directorio en una lista de directorios.
- **poll**: Duerme, esperando por actividad en un archivo.
- **ioct**: Envía un comando a un controlador de un dispositivo.
- **unlocked_ioctl**: Realiza la misma función que **ioct** pero permitiendo al controlador determinar el tipo de bloqueo que va a utilizar.
- **mmap**: Mapea en memoria un archivo recibido.
- **flush**: El propósito de esta función depende del sistema de ficheros. Es llamado por el sistema cuando la actividad sobre un archivo abierto decrece.
- **release**: Esta función es llamada por el Sistema cuando la última referencia a un achivo desaparece. Al igual que en el caso anterior, su comportamiento depende del sistema de ficheros:
- **fsync**: Escribe todos los datos cacheados referentes a un archivo a disco.
- **fasync**: Activa o desactiva la notificación de operaciones asíncronas.
- **lock**: Bloquea un archivo.
- **readv**: Lee de un archivo y guarda los datos en un vector de *buffers*.
- **writev**: Lee datos de un vector de *buffers* y los vuelca en un archivo.
- **sendfile**: Copia datos de un archivo a otro.
- **sendpage**: Envía datos de un archivo a otro.
- **get_unmapped_area**: Utiliza direcciones sin utilizar para mapear un archivo.
- **check_flags**: Comprueba la validez de un conjunto de flags.

- **flock**: Esta función espera hasta que un archivo pueda ser bloqueado para hacerlo.

2.3.5 Relación de los objetos

Ahora que, en las secciones anteriores, se han descrito las diferentes estructuras de datos importantes para el *VFS*, es conveniente mencionar la estructura jerárquica a la que obedecen.

En la parte superior se encontrarían los objetos de tipo *file*. Estos son directamente referenciados por la lista de descriptores de procesos y es con los que el *VFS* trata directamente. No es de extrañar que, como se describe en la sección anterior, las funciones implementadas por File se parezcan en su nombre y signatura a llamadas del sistema como *read()* o *write()*. De hecho, muchos de estos métodos son llamados directamente por el sistema. A parte de *read()* y *write(),* la llamada del sistema *mkdir()* llama directamente a la función *mkdir* de un objeto de tipo File para crear un directorio. Existen más casos similares cómo *readdir*, *open*, *flock*, etcétera…

Un nivel por debajo se encontrarían los objetos de tipo *dentry* y bajo estos, los de tipo *inodo*. Ambos hacen referencia a un objeto de tipo *superbloque*, que se encontraría en el nivel más bajo.

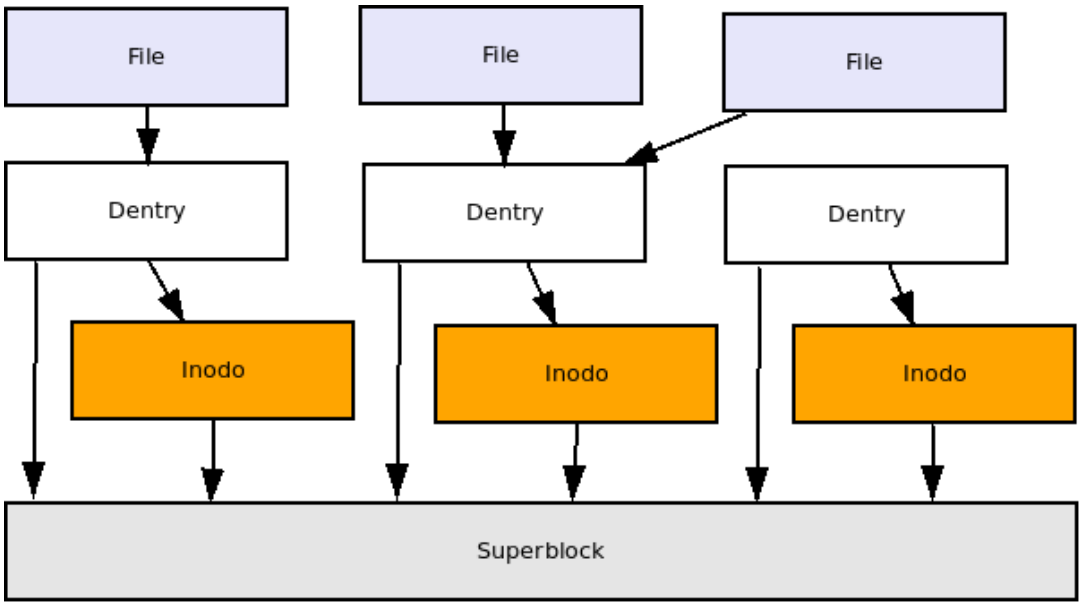


Ilustración 3 File, Dentry, Inodo y Superbloque

Como se puede observar en la figura, un objeto de tipo *dentry* puede hacer referencia a otro de mismo tipo. En cuanto a los objetos de tipo File, sólo pueden referenciar a objetos de tipo

dentry sin embargo, no hay restricciones en cuanto a cuantos objetos de tipo *file* pueden referenciar a uno de tipo *dentry*.

2.4 Sistemas de fichero en espacio de usuario

Al hablar de sistemas de fichero en espacio de usuario, nos referimos a un mecanismo que permite a usuarios no privilegiados, crear sus propios sistemas de ficheros sin modificar el código del *kernel*.

Esta funcionalidad, proporcionada, en un principio por sistemas operativos basados en Unix, otorga al usuario la posibilidad de definir sus propias reglas de acceso y lectura a los sistemas de ficheros creados en su espacio.

Ventajas e inconvenientes

Desarrollar sistemas de ficheros en el espacio de usuario tiene varias ventajas. La primera y primordial de cara al propio desarrollo es que el conjunto de funciones generadas es menos propenso a fallos graves. Los errores suelen producirse menos frecuentemente y, debido a que se ejecuta en espacio de usuario, las consecuencias de los mismos están mucho más controladas.

Siguiendo el mismo punto de vista, el desarrollo en espacio de usuario nos permite usar de forma libre una gran variedad de librerías que pueden proporcionar, desde estructuras de datos para simplificar el trabajo hasta funcionalidades enteras. Además, el código generado es más limpio y más fácil de testear

Como contrapartida, los sistemas de ficheros en espacio de usuario tienen un rendimiento peor. Por lo que si se pretende generar un sistema de ficheros de alto rendimiento, sería más conveniente desarrollarlo en el espacio del *kernel*.

A continuación, se describirá *FUSE*, la herramienta solicitada por el cliente para implementar el sistema de ficheros. Además, se describirá brevemente una de sus alternativas para *NetBSD*, *Puffs*.

2.4.1 FUSE

FUSE es un módulo del *kernel* de *Linux* que permite a los usuarios sin permisos de administrador, desarrollar sistemas de ficheros. Para ello, requiere que el programador implemente una interfaz consistente en usa serie de operaciones sobre archivos.

Los sistemas de ficheros creados mediante la tecnología *FUSE* pueden ser montados de forma segura por usuarios sin privilegios dentro de su espacio de usuario. Además, la implementación de estos, no requiere la re-compilación del *kernel* ni su reinicio.

2.4.1.1 Funcionamiento

Una vez montado el volumen, todas las llamadas del sistema que apuntan al punto de montajes son delegadas a las librerías de *FUSE* y estas, registran el volumen en el *VFS* con el tipo *fusefs*.

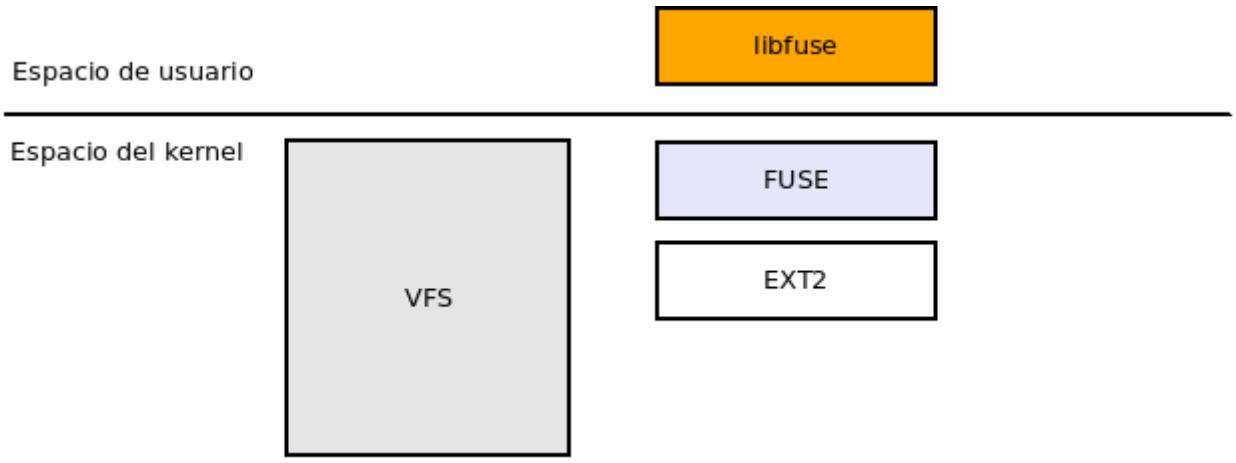


Ilustración 4 Estructura básica FUSE

La funcionalidad en el espacio de usuario será implementada a través de una serie de funciones de *callback*, que serán empleadas para ser utilizadas por *FUSE* en el punto de montaje.

Pongamos como ejemplo que el directorio */tmp/fuse* es el punto de montaje. Cuando una aplicación efectúa una llamada al sistema de tipo *read()* sobre, por ejemplo, /tmp/fuse/some_file.txt, el sistema virtual de ficheros accede al módulo de *FUSE* donde, utilizándo la librería *libfuse* accede a la función de *callback* correspondiente a la llamada del sistema. Esta función ejecutará las acciones necesarias y devolverá los datos en un buffer. Las acciones tomadas al resolver la llamada *read()* serán dependientes del tipo sistema de ficheros en el que se encuentren los datos que se desean leer. En última instancia, el resultado será enviado, por medio de *libfuse* y a través del *kernel* a la aplicación que ejecutó la llamada al sistema *read().*

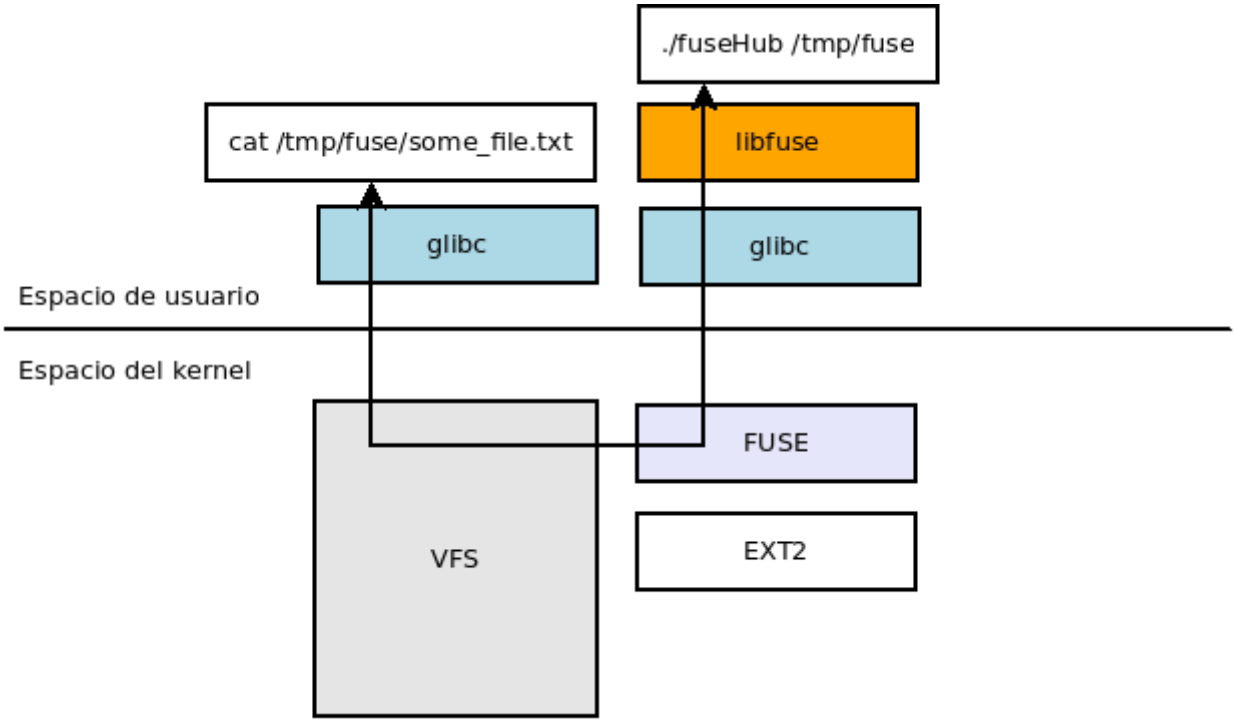


Ilustración 5 Ejemplo FUSE

Los pasos para realizar el flujo definido anteriormente pueden ser desarrollados a través de dos librerías ofrecidas por *FUSE*. Por un lado, provee un un *API* de alto nivel en el que el sistema desarrollado sólo necesita trabajar con *pathnames*. En este tipo de sistemas, es la librería *libfuse* la encargada de realizar las transformaciones de *inodo* a *pathname* y de propagar las estructuras de datos a través del *kernel*. Esta librería es la más utilizada para implementar sistemas de ficheros de tipo *FUSE* ya que cubre, de una forma sencillas, la mayor parte de la funcionalidad necesaria.

Como siguiente opción, *FUSE* incluye un *API* de bajo nivel que asemeja a la interfaz de *VFS*. Los sistemas implementados con estas librerías deben completar manualmente las estructuras de datos (*inodos*, principalmente) para dar funcionalidad el sistema de ficheros. Este *API* es muy útil cuando se pretende desarrollar un sistema de ficheros desde cero, es decir, cuando no se trata de añadir funcionalidad a una ya existente.

2.4.1.2 Lenguajes soportados

FUSE es un desarrollo escrito en el lenguaje de programación c, sin embargo, soporta más de 20 lenguajes de programación para implementar sistemas de ficheros. Algunos de ellos son:

- *C, C++ y C#*
- *Java*
- *Haskell*
- *Ruby*

- *Python*

2.4.2 Puffs

Puffs es un conjunto de librerías orientadas a la creación de sistemas de ficheros en espacio de usuario para *NetBSD*. Ofrece de un interfaz similar al de *VFS* a procesos del usuario acoplándose a la capa de este.

Su comportamiento básico, consiste en recibir peticiones del *kernel* a través del interfaz del VFS y enviarlas al espacio de usuario. De esta forma, ni el *kernel* ni ningún otro proceso, a parte del VFS, es capaz de distinguir un sistema de ficheros implementado en *puffs* de otro implementado en el *kernel*.

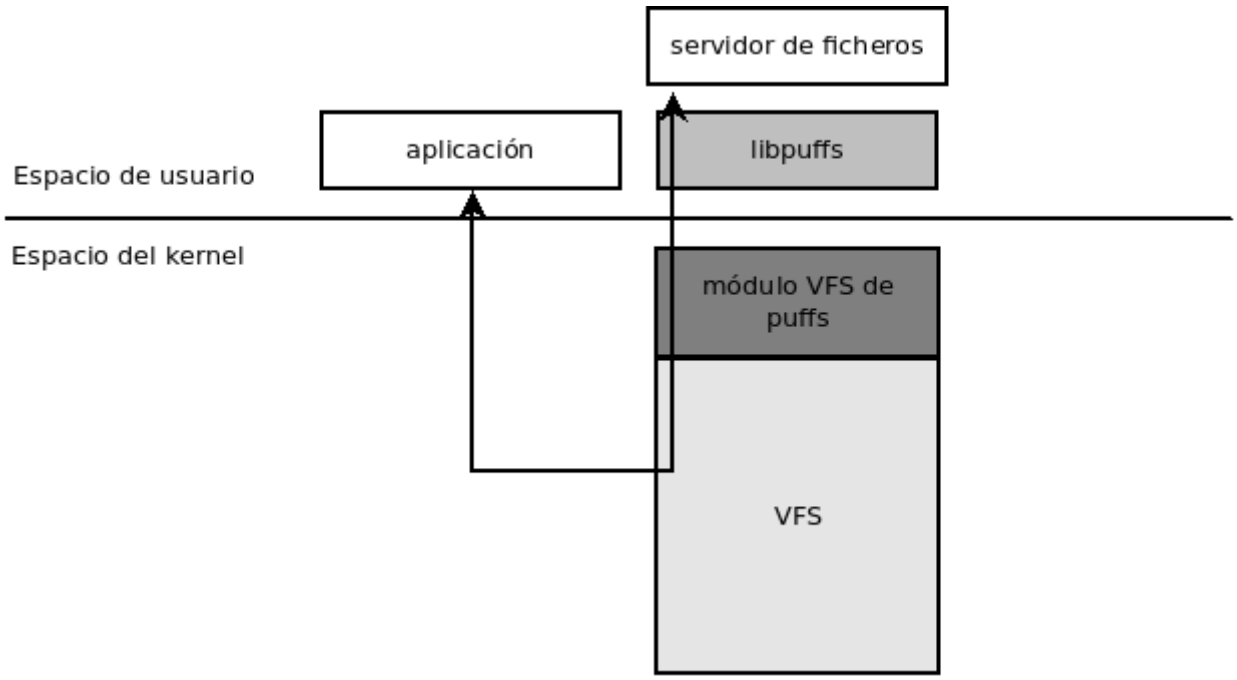


Ilustración 6 Ejemplo Puffs

De cara al sistema de ficheros, *puffs* expone una interfaz que se encuentra dividida en dos secciones, las operaciones sobre el sistema de ficheros y las operaciones sobre nodos.

Para el espacio de usuario se provee la librería *libpuffs* que, además de ofrecer una interfaz para desarrollar el sistema de ficheros, implementa una serie de funcionalidades comunes necesarias para el desarrollo.

2.4.2.1 Funcionamiento

El primer paso a seguir es registrar una serie de funciones de *callback* en *libpuffs* para montar el sistema de ficheros. En este punto, el control es cedido a la librería que será la encargada de recoger las peticiones realizadas en el especio de usuario y enviarlas al sistema de ficheros a través del *kernel*.

El API ofrecido consta de una serie de funciones de bajo nivel en las que hay un uso mínimo de *pathnames*. En su lugar, *puffs* utiliza su propia abstracción, llamada *vnode*. El problema reside en que la mayor parte de las operaciones sobre sistemas de ficheros requieren de *pathname*. Para solucionarlo *puffs* cuenta con una serie de puntos de acople y opciones de montado en los que el sistema puede registrar sus funciones para manejar *pathnames*.

Esta decisión en el diseño permite evitar confusiones a la hora de manejar varios links que apuntan a un mismo fichero. Como segunda ventaja, se consigue que las operaciones de renombrado sean más baratas, ya que no es necesario renombrar todos los *inodos* de un directorio.

2.4.2.2 Lenguajes soportados

Al igual que FUSE, *puffs* está implementado en C y permite el desarrollo de nuevos módulos en este lenguaje.

2.5 Tecnología utilizada

Una vez expuestos los conceptos y herramientas necesarios para el desarrollo del sistema, quedan establecidas las bases para el análisis y el posterior desarrollo.

En los siguientes apartados, se enumerarán una serie de herramientas que serán utilizadas durante el desarrollo. Para cada una de ellas, se expondrá un breve resumen en el que se mencionarán sus características principales.

2.5.1 Herramientas básicas

A continuación, se muestra un listado de las herramientas y utilidades que se utilizarán para el desarrollo del proyecto.

- **Entorno de desarrollo: *Eclipse for C/C++ developers*.**

  Eclipse es un entorno de desarrollo integrado por un conjunto de herramientas de código abierto.

  Se trabajará con la versión específica para desarrollos en *C/C++* ya que cuenta con una serie de funcionalidades integradas que permiten el desarrollo ágil y el control de versiones.

- **Lenguaje de programación: *C***

C es un lenguaje de programación imperativo, de propósito general que soporta programación estructurada.

Es el lenguaje más popular en el desarrollo de componentes *FUSE* y existen gran cantidad de librerías y ejemplos describiendo el uso de las llamadas del sistema que serán necesarias para el desarrollo del sistema.

La versión utilizada será C11.

- **Compilador: gcc**

Gcc es un compilador perteneciente al proyecto *GNU*. Es distribuido por la *Free Software Foundation* bajo licencia *GPL*.

Es considerado un estándar y parte esencial del proyecto *GNU* ya que, además de *C*, soporta la compilación de otros lenguajes como *C++, Fortran* o *Ada*.

Se utilizará la versión 5.2.

- **Control de versiones: *Git***

Se trata de un software de control de versiones muy popular, distribuido bajo licencia *GPL*.

Git está orientado hacia el desarrollo no lineal de código fuente. A diferencia de subversión, *Git* permite a los usuarios mantener ramas locales del código. Esto facilita la gestión del fusionado de diferentes ramas y ayuda a mantener código no comprometido en el servidor.

Se utilizará la versión 2.4.6.

2.5.2 Herramientas específicas

Para el desarrollo de proyecto, el cliente ha impuesto el uso de la librería *FUSE* en su versión 2.9.3 para la implementación del sistema de ficheros.

Al montar varios sistemas de ficheros diferentes en el mismo punto de montaje, se van a necesitar las librerías específicas para cada uno de ellos de las que dispone el *kernel*. Cómo se exponía en secciones anteriores, este dispone del *VFS* cómo capa de abstracción para interactuar con los sistemas de ficheros. Aprovechando esta capacidad, se utilizarán las librerías

de alto nivel de *FUSE* para realizar llamadas de sistema para que el *VFS* las resuelva, y se delegará la elección del sistema de ficheros al sistema desarrollado.

2.6 Metodología

2.6.1 Scrum

Scrum es un marco de trabajo que abarca una serie de buenas prácticas relacionadas con el trabajo en equipo. *Scrum* es un modelo que provee de un conjunto de prácticas y roles para ser tomadas como punto de partida para definir el proceso de desarrollo que se ejecutará durante el proyecto.

En la práctica, se aplican estrategias de desarrollo incremental en bloques temporales cortos y fijos, basando el resultado en la calidad del producto obtenido. De esta forma, se solapan las diferentes fases del desarrollo en lugar de realizarlas una tras otra como se realizaría en un proceso secuencial o en cascada.

Esta metodología es flexible ante cambios en los requisitos o la falta de una definición completa de los mismos, durante el proceso de desarrollo. Además, permite realizar un control empírico del proyecto. Por un lado, al final de cada iteración se muestran los avances obtenidos a la parte interesada en el proyecto, lo que permite que se puedan tomar decisiones en función de lo mostrado. Por otro, el equipo se mantiene informado diariamente de los avances realizados y de los posibles impedimentos encontrados, pudiendo de esta manera, tomar acciones al respeto de una forma mucho más rápida y eficiente.

La aplicación de *Scrum* en proyectos permite reducir el *Time To Market* ya que el cliente puede priorizar las características más importantes del proyecto para que sean realizadas en primer lugar y, así, empezar a poner en explotación el producto antes de estar completamente realizado.

Gracias a que se trabaja con espacios de tiempo cortos y fijos, una vez que el equipo ha alcanzado una velocidad de desarrollo estable (entre 3 y 5 iteraciones tras el inicio del proyecto) resulta sencillo predecir el tiempo que se va a necesitar para implementar una funcionalidad que se encuentra en la lista de funcionalidades pendientes.

2.6.1.1 Roles

En esta sección se enumerarán los diferentes roles que, habitualmente, intervienen en un proyecto ágil desarrollado con *Scrum*.

- **Stakeholder**: Este rol representa a cualquiera que tenga interés en el proyecto. Puede ser el Jefe de Proyecto, inversores del producto o los responsables del equipo

de desarrollo. Su responsabilidad es la de comunicar y proveer *feedback* sobre el producto que se está desarrollando.

- **Product Owner**: Representa al cliente, los usuarios y resto de partes interesadas en el producto (*Stakeholders*). Es la única persona autorizada para decidir sobre las funcionalidades del producto. Sus funciones y responsabilidades son:

    - Maximizar el valor de negocio a través del equipo
    - Mantener y priorizar el *Product Backlog*.
    - Crear y mantener el *Release Burndown Chart*
    - Ayudar al Scrum Master a organizar las reuniones de revisión del sprint.
    - Comunicar de forma clara las necesidades del negocio al equipo y a los *Stakeholders*.
    - Mantener un canal de comunicación con los *Stakeholders*.
    - Comunicar el progreso a los *Stakeholders* regularmente.
    - Asegurar el correcto uso de los recursos y ayudar al equipo a obtener los recursos que necesite.
    - Acudir a todas las reuniones del *SCRUM*.

- **Scrum Master**: Es el encargado de fomentar e instruir sobre los principios ágiles de *SCRUM*. Sus funciones son las siguientes
    - Garantizar la correcta aplicación de *SCRUM*.
    - Organizar los *Sprint Plannings*, *Daily Scrums*, *Sprint Reviews* y retrospectivas.
    - Crear el tablón de tareas y el *Sprint Burndown Chart* al comienzo de cada *sprint*.
    - Mantener el foco del equipo.
    - Avisar al equipo de posibles bloqueos y ayudar a resolverlos.
    - Acudir a todas las reuniones del *SCRUM*.

- **Equipo de desarrollo**: Son los encargados de desarrollar el producto. Debe ser un equipo multidisciplinar integrado por programadores, diseñadores, testers y demás. Sus funciones son:
    - Llegar a los objetivos establecidos para el Sprint.
    - Descomponer los elementos del *Backlog*.
    - Ayudar a crear y mantener el *Sprint Backlog*, el *Burndown Chart* y el tablón de tareas.

- Enseñar el producto realizado al final de cada *Sprint*.
- Realizar las acciones determinadas en cada retrospectiva.
- Compartir conocimiento y experiencia con el resto de los componentes del equipo.
- Ayudar, en caso necesario, a cualquier miembro del equipo.
- Acudir a todas las reuniones del *SCRUM*.

2.6.1.2 Sprints

Un sprint es la unidad básica de desarrollo en *SCRUM*. Se trata de periodo de tiempo fijo dedicado a realizar una serie de tareas planificadas. Su duración viene definida de antemano y nunca puede cambiarse una vez se ha comenzado el sprint.

Cada Sprint realizado comenzará con una reunión en la que establecerán los *PBIs* que van a realizarse (*Sprint Backlog*) y terminará con un una reunión para revisar lo realizado en el sprint y una retrospectiva.

Planificación del Sprint.

Se trata de una reunión programada al comienzo de cada sprint en la que se realizarán las siguientes tareas:

- Selección del trabajo que será realizado
- El equipo prepara el *backlog* dedicado al sprint que se va a comenzar. Durante esta fase, el equipo entero detallará el tiempo que será necesario para realizar el trabajo seleccionado en la fase anterior.
- Se redactará un acta comunicando cuanto del trabajo preseleccionado es posible realizar en el sprint que se va a comenzar.
- Se convocará al *product owner* y se determinará, junto con el equipo de desarrollo y el *SCRUM master* que *PBIs* van a ser implementados
- El equipo de desarrollo establecerá qué tareas son necesarias para realizar los *PBIs*.

Reuniones Stand-Up

Cada día, a la misma hora, el equipo de desarrollo tendrá una breve reunión en la que, cada miembro del equipo, responderá a las siguientes preguntas:

- ¿Qué hice ayer?
- ¿Qué haré hoy?
- ¿Tengo algún bloqueo que no vaya a permitirme, a mí o al equipo, acometer los hitos establecidos para el sprint actual?

El objetivo de estas reuniones es mantener al equipo y al *SCRUM master* informado del estado actual del desarrollo por lo que las respuestas deben ser lo más claras y concisas posibles.

En el caso de que se detecte algún bloqueo durante alguna de estas reuniones, este debe ser recogido por el *SCRUM Master* y alguien debe hacerse responsable en trabajar en su resolución.

Es muy importante que, a estas reuniones, asista todo el equipo de desarrollo y este haya preparado las respuestas a las preguntas expuestas anteriormente. Igualmente importante es que no se entre en discusiones sobre los detalles de las tareas realizadas, que se van a realizar o los bloqueos.

La reunión debería realizarse en el mismo lugar y a la misma hora cada día. Además, su duración no debería ser mayor a 15 minutos.

Reuniones de refinamiento

En estas reuniones cada requisito y error de la *Pila de Producto* es estimado o reestimado. Estas estimaciones se realizarán, de forma colaborativa, por todos los miembros del equipo de desarrollo y su valor estará expresado en puntos de historia.

Demo

Al final de cada Sprint se realizará una reunión para mostrar y revisar las historias de usuario que han sido completadas en la última iteración. Estas reuniones están abiertas a cualquiera que esté interesado en evaluar los resultados obtenidos. Es un buen momento para que los *Stakeholder* vean el progreso logrado y provean de *feedback* al *Product Owner*.

La demo debe centrarse en demostrar lo que el equipo de desarrollo ha realizado en el último sprint. La preparación corre a cargo del *Product Owner* y el equipo de desarrollo por lo que es importante que todos los *PBIs* completados en el Sprint, estén documentados y testeados antes de la reunión.

La presentación debe realizarse de forma clara, comenzando por una enumeración de los hitos a los que se pretendía llegar en el sprint y los que se han logrado, por parte del *Product Owner.* Seguidamente, el equipo de desarrollo demostrará cada uno de los *PBIs* completados durante el Sprint.

Los *Stakeholder* podrán, durante la demostración, realizar preguntas, solicitar aclaraciones o proveer de *feedback* sobre las funcionalidades implementadas.

Estas reuniones proveen al equipo de desarrollo la posibilidad de mostrar directamente el producto a los *Stakeholders* y recibir reconocimiento por el trabajo realizado. Esto mejora la moral del equipo y lo motiva para mantener o incrementar el nivel de calidad en el producto.

Retrospectiva

Tras la demo, el equipo de desarrollo se reunirá con el *SCRUM Master* para tener una reunión de retrospectiva. En ella, los asistentes pondrán en común consideraciones sobre lo que ha ido bien durante el sprint, lo que ha ido mal y se propondrán medidas a tomar para solucionar los problemas surgidos.

Para alcanzar los objetivos de estas reuniones, los asistentes deben ser capaces de hablar libremente sobre los fallos y aciertos obtenidos durante el sprint. De esta forma, el equipo puede atajar problemas relacionados con su efectividad y productividad e identificar estrategias para mejorar el proceso.

Estas reuniones tienen una gran importancia para el *SCRUM Master*, ya que, le permiten observar bloqueos y problemas comunes que pueden afectar al equipo y al trabajo que lleva a cabo.

2.6.1.3 Estimaciones y la velocidad del equipo.

Puntos de historia

El proceso de estimación a través de puntos de historia busca determinar el tamaño de una Historia de usuario para que este sea igual para cualquier miembro del equipo.

Los puntos de historia son el tamaño relativo de una tarea en comparación con otras estimadas por el mismo equipo o, dicho con otras palabras, una medida para estimar el esfuerzo necesario para implementar una historia de usuario. Dicho esfuerzo puede estar dado en función de la complejidad de la historia de usuario o el desconocimiento sobre.

Velocidad del equipo

En un proyecto *SCRUM*, la velocidad obtenida durante una iteración se calcula sumando el número de puntos de historia de todos los *PBIs* terminados durante un sprint del proyecto.

Entre el tercer y el quinto *sprint*, la velocidad de desarrollo se estabiliza con lo que, utilizando un listado histórico de las velocidades por sprint, puede establecerse una correlación entre el punto de historia y el tiempo que tarda el equipo en desarrollarlo.

Hay que tener en cuenta que, debido a la diferencia de conocimiento y experiencia, la medida de tiempo necesario para implementar un punto de historia no es extrapolable a otros equipos, sólo es válida en el equipo que se ha calculado. Un equipo estable formado por tres programadores senior puede tener una velocidad por sprint de cuarenta puntos de historia mientras que un equipo recién formado y que sólo está compuesto por programadores junior puede alcanzar veinte puntos de historia por sprint.

2.6.1.4 Documentos

Pila de producto o Product Backlog

El *Product Backlog* es un documento de análisis muy detallado en el que se muestran, en forma de lista priorizada, todos los requerimientos del proyecto. La prioridad de los elementos que la componen vendrá dictada por el *Product Owner* en función del valor de negocio que aporte.

Este documento actúa como motor del negocio, partiendo las funcionalidades de más alto nivel en incrementos de trabajo más manejables, los *Product Backlog Items* (*PBIs* a partir de ahora).

Pila de la iteración o Sprint Backlog

Al comienzo de un sprint, el *Product Owner* selecciona una serie de *PBIs* del *Product Backlog* para que sean implementados en la iteración actual. El equipo divide estos *PBIs* en tareas más sencillas que describen las funcionalidades que componen el proyecto. A este listado se le denomina *Sprint Backlog*.

Las tareas que componen el *Sprint Backlog* deben ser pequeñas y poco acopladas entre ellas para poder ser estimadas. Además, es el equipo de desarrollo, debido a que es el que tiene el conocimiento sobre la arquitectura del sistema, el que decide cuáles, en qué orden y de qué forma se realizarán estas tareas.

2.6.1.5 Plan de Liberación y Mínimo Producto Viable

Para ayudar al *product owner* y al equipo a decidir cuánto producto debe desarrollarse y cuánto tiempo se tardará antes de tener un producto entregable, se utilizan los planes de liberación.

Un plan de liberación es un conjunto de PBIs agrupadas por liberaciones o *releases*. Comunmente, el versionado de la aplicación se incrementa en cada una de estas liberaciones.

El concepto de *release* ayuda al equipo a enfocar su esfuerzo en la consecución de un objetivo marcado en un plazo de tiempo realizable y cercano y evita moverse sin rumbo de una iteración a otra.

Los *PBIs* incluidos en cada reléase deben estar orientados para alcanzar un mínimo producto viable o MVP, una versión del producto que permita al equipo conseguir el máximo de las necesidades reales del *product owner* sobre el producto en el mínimo tiempo.

El compromiso adquirido por el equipo en cada una de las *releases* no es con la fecha de entrega, sino con la prioridad marcada por el *product owner* y la búsqueda de un ritmo de desarrollo sostenido y óptimo.

Habitualmente, las *releases* necesitán un número diferente de iteraciones para ser completadas pero debe refinarse el producto hasta que sean lo más pequeñas posibles.

2.7 Conclusiones Sobre la Metodología

Ventajas e Inconvenientes

El uso de marcos de trabajo ágiles como *SCRUM* resulta beneficioso para las empresas y trabajadores ya que permite, gracias a los procesos frecuentes de revisión y validación, mayor velocidad de movimiento y reacción ante cambios en el alcance del proyecto y errores. La detección y corrección de estos problemas resulta mucho más sencilla para el equipo.

SCRUM, en concreto, aunque también es aplicable al resto de metodologías ágiles, es un proceso muy controlado y orientado al trabajo en equipo gracias a las reuniones frecuentes que se realizan. Gracias a ellas, los bloqueos, desviaciones y avance del proyecto son fácilmente detectados.

Gracias a la implicación del *product owner* en el proceso, resulta más sencillo detectar las necesidades reales y su prioridad para el avance del proyecto. Es el *product owner* el que decide, gracias a la priorización de los *PBIs*, que funcionales resultan críticas y cuáles pueden ser consideradas en otro punto del desarrollo.

En el lado contrario, la falta de una definición clara puede suponer problemas en el proceso de planificación y organización del proyecto. Además, los cambios frecuentes de alcance o la falta de claridad a la hora de definir objetivos pueden desestabilizar el proceso de desarrollo. Este tipo de problemas suelen derivarse de la falta de implicación de alguno de los actores que toman parte en SCRUM.

Conclusiones

El sistema que se va a desarrollar tiene un conjunto de funcionalidades muy definidas, cuyo desarrollo puede ser dividido y entregado en diferentes *releases* para su prueba, evaluación y puesta en producción.

Cada reléase, será probada delante del *product owner* que validará el comportamiento e informará sobre posibles modificaciones y discrepancias. Esto permitirá al equipo realizar un trabajo mucho más cercano a los intereses adquiridos en el sistema y lo dotará de un mejor tiempo de reacción de cara a modificaciones y errores.

Como se ha explicado anteriormente, una de las mejores características que aporta *SCRUM* al proceso de desarrollo, es la flexibilidad ante las modificaciones en el alcance tanto de *PBIs*, cómo de nueva funcionalidad. La adopción de *SCRUM* por parte del equipo facilitará y mejorará el proceso de generación del sistema y, además permitirá una transición más sencilla y fluida hacia las siguientes fases de desarrollo donde, se incluirá nueva funcionalidad y podría solicitarse modificaciones sobre el comportamiento actual.

3 Análisis

3.1 Requisitos de Usuario

En la siguiente sección se enumeran los requisitos de usuario que deben estar implementados en la solución.

Para simplificar su lectura, seguirán en siguiente formato:

Código Identificador			
Descripción			
Necesidad		**Origen**	
Variabilidad		**Verificabilidad**	

Tabla 1 Formato requisitos de usuario

- **Código identificador**: Es un código único que identifica a cada requisito. Sigue el formato **UR_XX_YY**, donde se distingue:
  - **UR**: Responde a las siglas de **User Requirement**.
  - **XX**: Hace referencia al tipo de requisito de usuario que se va a describir. Puede tener los siguientes valores:
    - **RE**: Requisito de restricción
    - **CA**: Requisito de capacidad
  - **YY**: Número de requisito. Se calcula en función de los valores asignados con anterioridad para las categorías definidas en el nivel anterior de indexación.

- **Descripción**: Detalle preciso y simple sobre el requisito de usuario.

- **Necesidad**: Establece la importancia del requisito dentro del desarrollo. Puede tomar los siguientes valores: **Baja**, **Media** o **Alta**.

- **Variabilidad**: Establece el grado de cambio que puede sufrir el requisito a lo largo del desarrollo del proyecto. Puede tomar los valores **Estable** o **Variable**.

- **Origen**: Establece, en función de su valor, el origen del requisito pudiendo ser por parte del cliente (**Cliente**) o por parte del equipo de desarrollo del proyecto (**Equipo**).

- **Verificabilidad**: Mide la dificultad que posee la verificación del cumplimiento del requisito. Puede tomar los valores **Baja**, **Media** o **Alta**.

3.1.1 Requisitos de capacidad

CA_UR_01			
Descripción	El usuario debe poder listar los atributos de un fichero o carpeta que sea accesible desde el punto de montaje.		
Necesidad	Alta	**Origen**	Equipo
Variabilidad	Estable	**Verificabilidad**	Alta

Tabla 2 CA_UR_01

CA_UR_02			
Descripción	El usuario podrá crear carpetas dentro del punto de montaje seleccionado.		
Necesidad	Alta	**Origen**	Cliente
Variabilidad	Estable	**Verificabilidad**	Alta

Tabla 3 CA_UR_02

CA_UR_03			
Descripción	El usuario podrá acceder a las carpetas creadas dentro del punto de montaje seleccionado.		
Necesidad	Alta	**Origen**	Cliente
Variabilidad	Media	**Verificabilidad**	Media

Tabla 4 CA_UR_03

CA_UR_04			
Descripción	El usuario podrá borrar carpetas que sean accesibles dentro del punto de montaje seleccionado.		
Necesidad	Media	**Origen**	Cliente
Variabilidad	Media	**Verificabilidad**	Media

Tabla 5 CA_UR_04

CA_UR_05			
Descripción	El usuario podrá listar el contenido de las carpetas que se encuentren dentro del punto de montaje seleccionado		
Necesidad	Alta	**Origen**	Cliente
Variabilidad	Baja	**Verificabilidad**	Baja

Tabla 6 CA_UR_05

CA_UR_06			
Descripción	El usuario podrá crear ficheros nuevos dentro de la estructura de carpetas alojada dentro del punto de montaje		
Necesidad	Alta	**Origen**	Cliente
Variabilidad	Alta	**Verificabilidad**	Media

Tabla 7 CA_UR_06

CA_UR_07			
Descripción	El usuario podrá copiar ficheros localizados fuera de la estructura alojada en el punto de montaje a una de las carpetas que se encuentren dentro del punto de montaje		
Necesidad	Media	**Origen**	Cliente

CA_UR_07			
Variabilidad	Alta	**Verificabilidad**	Media

Tabla 8 CA_UR_07

CA_UR_08			
Descripción	El usuario podrá abrir los ficheros alojados en la estructura de carpetas que se encuentra dentro del punto de montaje		
Necesidad	Alta	**Origen**	Cliente
Variabilidad	Media	**Verificabilidad**	Alta

Tabla 9 CA_UR_08

CA_UR_09			
Descripción	El usuario podrá renombrar ficheros nuevos dentro de la estructura de carpetas alojada dentro del punto de montaje		
Necesidad	Baja	**Origen**	Cliente
Variabilidad	Baja	**Verificabilidad**	Alta

Tabla 10 CA_UR_09

CA_UR_10			
Descripción	El usuario podrá montar varios sistemas de ficheros en un punto de montaje.		
Necesidad	Alta	**Origen**	Cliente
Variabilidad	Baja	**Verificabilidad**	Alta

Tabla 11 CA_UR_10

CA_UR_11			
Descripción	El usuario recibirá información si no se cumplen los requisitos básicos para el montaje de los sistemas de ficheros.		
Necesidad	Alta	**Origen**	Cliente
Variabilidad	Baja	**Verificabilidad**	Alta

Tabla 12 CA_UR_11

3.1.2 Requisitos de Restricción

RE_UR_01			
Descripción	El montaje de las diferentes carpetas se realizará mediante la línea de comandos		
Necesidad	Alta	**Origen**	Equipo
Variabilidad	Baja	**Verificabilidad**	Alta

Tabla 13 RE_UR_01

RE_UR_02			
Descripción	El sistema permitirá montar un mínimo de una carpeta y un máximo de cinco.		
Necesidad	Alta	**Origen**	Equipo
Variabilidad	Baja	**Verificabilidad**	Alta

Tabla 14 RE_UR_02

RE_UR_03	
Descripción	El sistema debe guardar trazas sobre los eventos ocurridos en el punto de montaje en un fichero de texto.

RE_UR_03			
Necesidad	Alta	**Origen**	Equipo
Variabilidad	Media	**Verificabilidad**	Alta

Tabla 15 RE_UR_03

RE_UR_04			
Descripción	El sistema creará los ficheros y carpetas con permisos de lectura, escritura y ejecución.		
Necesidad	Alta	**Origen**	Equipo
Variabilidad	Baja	**Verificabilidad**	Alta

Tabla 16 RE_UR_04

RE_UR_05			
Descripción	El sistema permitirá montar los sistemas de ficheros de tipo : • EXT 2 • EXT 3 • EXT 4 • XFS • JFS		
Necesidad	Alta	**Origen**	Cliente
Variabilidad	Baja	**Verificabilidad**	Alta

Tabla 17 RE_UR_05

3.2 Casos de Uso

Los casos de uso describen el conjunto de situaciones en las que puede encontrarse un usuario mientras utiliza la aplicación. Cada caso de uso enumera los pasos que debe seguir un actor (en este caso, el usuario del sistema) para lograr aplicar una determinada funcionalidad.

Para la descripción de cada uno de los casos de uso se utilizará la siguiente plantilla:

<table>
<tr><th>Código Identificador</th><th>Nombre</th></tr>
<tr><td>Dependencias</td><td></td></tr>
<tr><td>Precondiciones</td><td></td></tr>
<tr><td>Secuencia</td><td><table><tr><th>Paso</th><th>Descripción</th></tr><tr><td>1</td><td></td></tr></table></td></tr>
<tr><td>Postcondiciones</td><td></td></tr>
<tr><td>Excepciones</td><td></td></tr>
</table>

Tabla 18 Formato casos de uso

- **Código Identificador**: Código único que identifica a cada caso de uso. Sigue el formato **UC_XX** donde:
  - **UC**: Corresponde a las siglas de User Case.
  - **XX**: Numerador único para el caso de uso

- **Nombre**: Descripción breve y precisa del caso de uso. Este campo será el utilizado en el gráfico de casos de uso.

- **Dependencias**: Otros casos de uso necesarios para la implementación de actual.

- **Precondiciones**: Lista de condiciones necesarias para la ejecución básica del caso de uso.

- **Secuencia**: Lista de eventos correspondientes a la ejecución del caso de uso. La plantilla utilizada para describir esta lista está compuesta por:
    - **Paso**: Orden del paso descrito.
    - **Descripción**: Texto descriptivo, conciso y simple sobre el paso que debe llevarse a cabo.

- **Postcondiciones**: Listado de eventos posteriores a la ejecución del caso de uso.

- **Excepciones**: Posibles excepciones ocurridas durante la ejecución del caso de uso.

3.2.1 Carpetas

En esta sección se describen las interacciones que puede mantener el usuario con respecto a las carpetas. A continuación, se muestra un gráfico con los casos de uso y una descripción de los mismos.

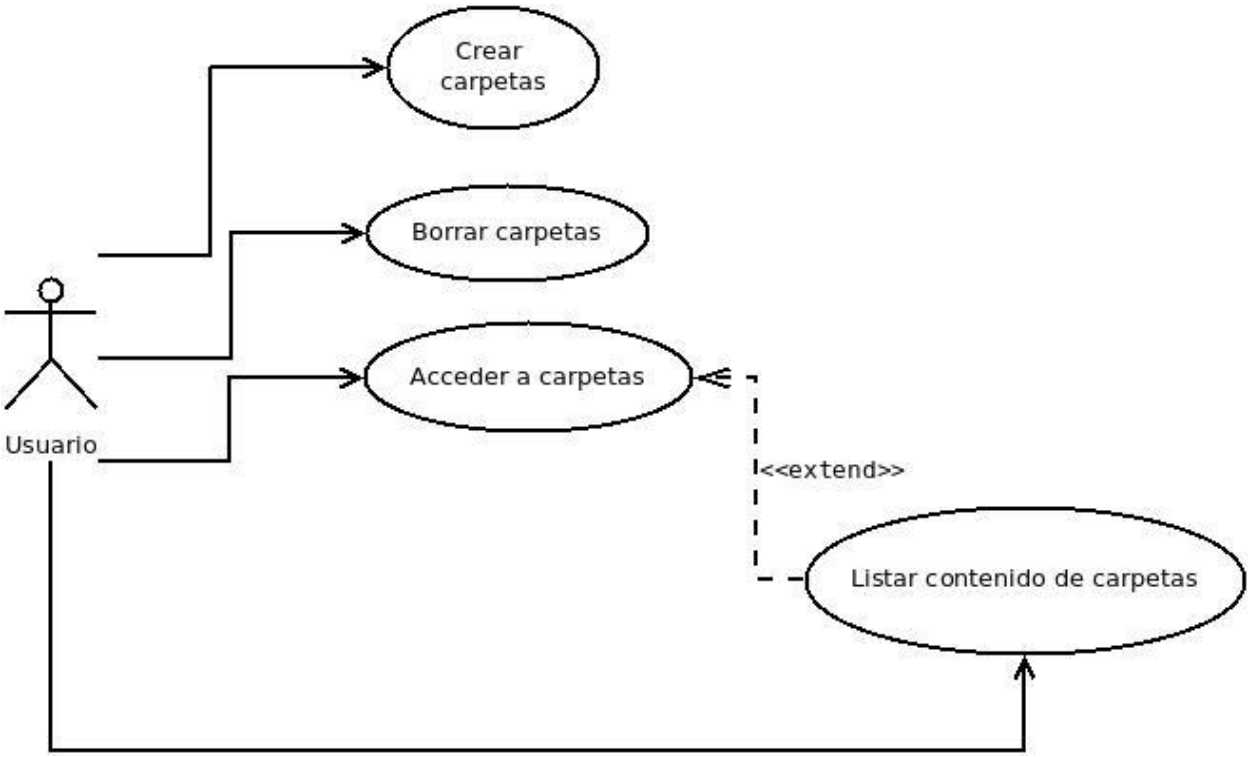


Ilustración 7 Casos de uso con carpetas

CU_01	Crear carpetas
Dependencias	No Aplica
Precondiciones	• La ruta de creación de la carpeta debe ser relativa al punto de montaje. • La ruta en la que se crea la carpeta debe existir con antelación.
Secuencia	**Paso** / **Descripción** **1** / El usuario introduce el comando de creación de carpetas

CU_01	Crear carpetas
Postcondiciones	• La carpeta es creada dentro de uno de los sistemas de ficheros montados en el punto de montaje. • La ruta de la carpeta creada en el sistema de ficheros será la misma con la que se creó en el punto de montaje.
Excepciones	No Aplica

Tabla 19 CU_01

CU_02	Acceder a carpetas
Dependencias	No Aplica
Precondiciones	• La carpeta a la que se pretende acceder debe haber sido creada con anterioridad dentro de alguno de los sistemas de ficheros montados en el punto de montaje. • La ruta de la carpeta debe ser relativa al punto de montaje.
Secuencia	**Paso** | **Descripción** **1** | El usuario introduce el comando de acceso a una carpeta.
Postcondiciones	• La ruta en la que se encuentra el usuario es relativa al punto de montaje.
Excepciones	No Aplica

Tabla 20 CU_02

CU_03	Borrar carpetas
Dependencias	No Aplica

<table>
<tr><th>CU_03</th><th>Borrar carpetas</th></tr>
<tr><td>Precondiciones</td><td>• La carpeta debe que se pretende borrar debe haber sido creada con anterioridad dentro de alguno de los sistemas de ficheros montados en el punto de montaje
• La ruta de la carpeta debe ser relativa al punto de montaje.</td></tr>
<tr><td>Secuencia</td><td><table><tr><th>Paso</th><th>Descripción</th></tr><tr><td>1</td><td>El usuario ejecuta un comando de borrado de carpetas</td></tr></table></td></tr>
<tr><td>Postcondiciones</td><td>• La carpeta es borrada de todos los sistemas de ficheros montados en el sistema de ficheros.</td></tr>
<tr><td>Excepciones</td><td>No Aplica</td></tr>
</table>

Tabla 21 CU_03

<table>
<tr><th>CU_04</th><th>Listar contenido de carpetas</th></tr>
<tr><td>Dependencias</td><td>• CU-02</td></tr>
<tr><td>Precondiciones</td><td>• La carpeta de la que se quiere mostrar el contenido debe haber sido creada con anterioridad en alguno de los sistemas de ficheros.</td></tr>
<tr><td>Secuencia</td><td><table><tr><th>Paso</th><th>Descripción</th></tr><tr><td>1</td><td>El usuario ejecuta un comando de listado de contenidos de una carpeta</td></tr></table></td></tr>
<tr><td>Postcondiciones</td><td>• Se muestra el contenido de la carpeta</td></tr>
<tr><td>Excepciones</td><td>No Aplica</td></tr>
</table>

Tabla 22 CU_04

3.2.2 Ficheros

A continuación, se muestran las interacciones posibles del usuario con los ficheros:

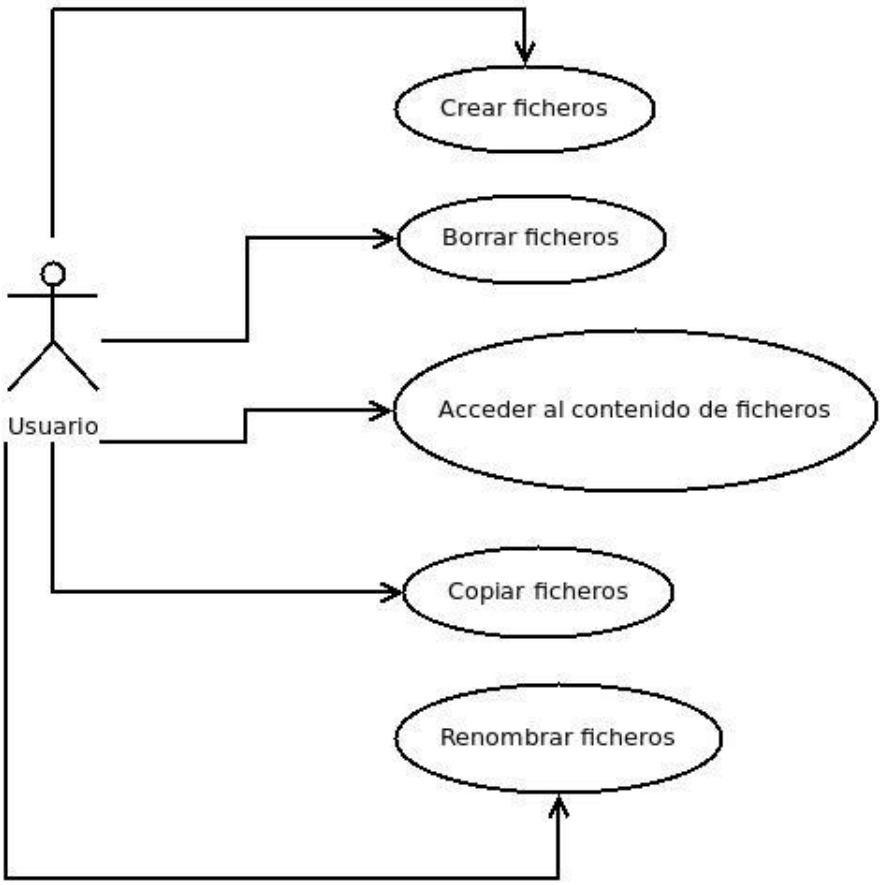


Ilustración 8 Casos de uso con ficheros

CU_05	Crear ficheros
Dependencias	No Aplica
Precondiciones	• La ruta de creación del fichero debe ser relativa al punto de montaje. • La ruta en la que se crea el fichero debe existir con antelación.
Secuencia	**Paso** / **Descripción** **1** / El usuario ejecuta un comando de creación de ficheros

CU_05	Crear ficheros
Postcondiciones	• El fichero es creado dentro de uno de los sistemas de ficheros montados en el punto de montaje. • La ruta del fichero creado en el sistema de ficheros será la misma con la que se creó en el punto de montaje.
Excepciones	No Aplica

Tabla 23 CU_05

CU_06	Copiar ficheros	
Dependencias	No Aplica	
Precondiciones	• La ruta a la que se quiere copiar el fichero debe ser relativa al punto de montaje • La ruta en la que se encuentra el fichero no debe ser relativa al punto de montaje. • La ruta a la que se quiere copiar el fichero debe existir con antelación.	
Secuencia	**Paso**	**Descripción**
	1	El usuario ejecuta un comando de copiado de ficheros
Postcondiciones	• El fichero se copia a uno de los sistemas de ficheros montados en el punto de montaje.	
Excepciones	No Aplica	

Tabla 24 CU_06

CU_07	Acceder al contenido de ficheros
Dependencias	No Aplica

<table>
<tr><th>CU_07</th><th>Acceder al contenido de ficheros</th></tr>
<tr><td>Precondiciones</td><td>• El fichero que se quiere abrir debe encontrarse dentro de los sistemas de ficheros montados en el punto de montaje.
• La ruta del fichero debe ser relativa al punto de montaje.</td></tr>
<tr><td>Secuencia</td><td><table><tr><th>Paso</th><th>Descripción</th></tr><tr><td>1</td><td>El usuario ejecuta un comando de apertura de ficheros.</td></tr></table></td></tr>
<tr><td>Postcondiciones</td><td>• Se muestra el contenido del fichero</td></tr>
<tr><td>Excepciones</td><td>No Aplica</td></tr>
</table>

Tabla 25 CU_07

<table>
<tr><th>CU_08</th><th>Borrar ficheros</th></tr>
<tr><td>Dependencias</td><td>No Aplica</td></tr>
<tr><td>Precondiciones</td><td>• El fichero que se quiere borrar debe encontrarse dentro de los sistemas de ficheros montados en el punto de montaje.
• La ruta del fichero debe ser relativa al punto de montaje.</td></tr>
<tr><td>Secuencia</td><td><table><tr><th>Paso</th><th>Descripción</th></tr><tr><td>1</td><td>El usuario ejecuta un comando de borrado de ficheros</td></tr></table></td></tr>
<tr><td>Postcondiciones</td><td>• El fichero se borra de los sistemas de ficheros montados en el punto de montaje</td></tr>
<tr><td>Excepciones</td><td>No Aplica</td></tr>
</table>

Tabla 26 CU_08

<table>
<tr><th>CU_09</th><th>Renombrar ficheros</th></tr>
<tr><td>Dependencias</td><td>No Aplica</td></tr>
<tr><td>Precondiciones</td><td>El fichero que se quiere renombrar debe encontrarse dentro de los sistemas de ficheros montados en el punto de montaje.
La ruta del fichero debe ser relativa al punto de montaje.
No debe existir un fichero con el nombre que se desea asignar dentro de la misma carpeta</td></tr>
<tr><td>Secuencia</td><td><table><tr><th>Paso</th><th>Descripción</th></tr><tr><td>1</td><td>El usuario ejecuta un comando de renombrado de ficheros</td></tr></table></td></tr>
<tr><td>Postcondiciones</td><td>El fichero se renombra
El fichero se mantiene dentro del mismo sistema de ficheros en el que se encontraba antes de la ejecución del comando.</td></tr>
<tr><td>Excepciones</td><td>No Aplica</td></tr>
</table>

Tabla 27 CU_09

3.3 Requisitos De Software

En este apartado, se muestra la especificación de los requisitos de software, obtenidos a partir de los requisitos de usuario mediante un análisis exhaustivo de las funcionalidades requeridas y sus restricciones.

Las funcionalidades que se analizan, corresponden al programa de adaptación, de desarrollo propio, aunque también se detallan ideas implementadas por el software de terceros, especificando este aspecto en todo momento. Sólo se recogen las funcionalidades de los sistemas de terceros relacionadas con la adaptación y su propósito, siendo coherente la idea explicada en los casos de uso.

Los requisitos de software se dividen en funcionales, especificando la funcionalidad del software, y en no funcionales, especificando el procedimiento y la manera de realizar la funcionalidad.

Se utilizará el siguiente formato para definir cada uno de los requisitos:

Código identificador			
Descripción			
Necesidad		**Verificabilidad**	
Prioridad		**Estabilidad**	
Origen			

Tabla 28 Formato Requisito de software

Donde:

- **Código Identificador**: Es un código único que identificará a cada uno de los requisitos. Seguirá el formato SR_XX_ZZ, donde:
  - SR: Son las siglas de *Software Requirement.*
  - XX: Se sustituirá por F en el caso de que sea un requisito funcional o por NF en el caso de que sea un requisito no funcional.
  - ZZ: Número del requisito.

- **Descripción**: Texto que detalla la especificación del requisito de forma sencilla y clara.

- **Necesidad**: Establece la necesidad del requisito desde el punto de vista del cliente. Puede tener los valores: Alta, Media y Baja.

- **Prioridad**: Describe la importancia del requisito en función del desarrollo del proyecto. Puede tomar los valores: Alta, Media y Baja.

- **Estabilidad**: Cuantifica la variabilidad que puede sufrir el requisito a lo largo del desarrollo. Puede tomar los valores: Alta, Media y Baja.

- **Verificabilidad**: Mide la dificultad para verificar el cumplimiento del requisito. Puede tomar los valores: Alta, Media y Baja.

- **Origen**: Indica de cual o cuales requisitos de usuario procede el requisito funcional.

3.3.1 Requisitos Funcionales

RF_SR_01			
Descripción	El sistema permitirá el montaje de sistemas de ficheros desde línea de comandos. Se ejecutará la aplicación con dos parámetros: • Ruta a un fichero de texto con las rutas de los sistemas de ficheros que se desean montar • Ruta del punto de montaje		
Necesidad	Alta	**Verificabilidad**	Alta
Prioridad	Alta	**Estabilidad**	Baja
Origen	RE_UR_10, CA_UR_11, RE_UR_01		

Tabla 29 RF_SR_01

RF_SR_02			
Descripción	En el caso de que los parámetros no cumplan el formato adecuado, el sistema informará de los errores y mostrará un texto indicando el formato correcto		
Necesidad	Media	**Verificabilidad**	Alta
Prioridad	Baja	**Estabilidad**	Alta
Origen	CA_UR_11		

Tabla 30 RF_SR_02

RF_SR_03	
Descripción	El sistema localizará el sistema de ficheros en el que se encuentra un fichero accesible a través del punto de montaje

RF_SR_03			
Necesidad	Alta	**Verificabilidad**	Media
Prioridad	Alta	**Estabilidad**	Baja
Origen	CA_UR_01, CA_UR_08		

Tabla 31 RF_SR_03

RF_SR_04			
Descripción	El sistema podrá mostrar los atributos de un fichero alojado en uno de los sistemas de ficheros accesibles a través del punto de montaje		
Necesidad	Media	**Verificabilidad**	Alta
Prioridad	Media	**Estabilidad**	Media
Origen	CA_UR_01, CA_UR_08		

Tabla 32 RF_SR_04

RF_SR_05			
Descripción	El sistema permitirá crear carpetas en los sistemas de ficheros montados a través del punto de montaje		
Necesidad	Alta	**Verificabilidad**	Alta
Prioridad	Alta	**Estabilidad**	Alta
Origen	CA_UR_02		

Tabla 33 RF_SR_05

RF_SR_06			
Descripción	El sistema mostrará los errores ocurridos durante la creación de carpetas en los sistemas de ficheros a través del punto de montaje.		
Necesidad	Media	**Verificabilidad**	Alta
Prioridad	Media	**Estabilidad**	Media
Origen	CA_UR_02		

Tabla 34 RF_SR_06

RF_SR_07			
Descripción	El sistema permitirá acceder a las carpetas alojadas dentro de los sistemas de ficheros montados en el punto de montaje		
Necesidad	Alta	**Verificabilidad**	Alta
Prioridad	Alta	**Estabilidad**	Baja
Origen	CA_UR_03		

Tabla 35 RF_SR_07

RF_SR_08			
Descripción	El sistema mostrará los errores ocurridos al acceder a carpetas alojadas dentro de los sistemas de ficheros montados.		
Necesidad	Media	**Verificabilidad**	Alta
Prioridad	Media	**Estabilidad**	Media
Origen	CA_UR_03		

Tabla 36 RF_SR_08

RF_SR_09			
Descripción	El sistema permitirá borrar carpetas alojadas dentro de los sistemas de ficheros montados en el punto de montaje		
Necesidad	Alta	**Verificabilidad**	Alta
Prioridad	Media	**Estabilidad**	Alta
Origen	CA_UR_04		

Tabla 37 RF_SR_09

RF_SR_10			
Descripción	El sistema mostrará los errores ocurridos al borrar a carpetas alojadas dentro de los sistemas de ficheros montados.		
Necesidad	Alta	**Verificabilidad**	Alta
Prioridad	Media	**Estabilidad**	Alta
Origen	CA_UR_04		

Tabla 38 RF_SR_10

RF_SR_11			
Descripción	El sistema podrá mostrar un listado de los contenidos de una carpeta alojada en uno o varios de los sistemas de ficheros accesibles a través del punto de montaje		
Necesidad	Alta	**Verificabilidad**	Alta
Prioridad	Alta	**Estabilidad**	Media
Origen	CA_UR_05		

Tabla 39 RF_SR_11

RF_SR_12			
Descripción	El sistema permitirá crear ficheros nuevos dentro de los sistemas de ficheros montados		
Necesidad	Alta	**Verificabilidad**	Media
Prioridad	Alta	**Estabilidad**	Media
Origen	CA_UR_06		

Tabla 40 RF_SR_12

RF_SR_13			
Descripción	El sistema informará de los errores sucedidos en el proceso de creación de ficheros nuevos en los sistemas de ficheros.		
Necesidad	Media	**Verificabilidad**	Alta
Prioridad	Media	**Estabilidad**	Media
Origen	CA_UR_06		

Tabla 41 RF_SR_13

RF_SR_14			
Descripción	El sistema localizará el sistema de ficheros en el que se deben crear los nuevos ficheros.		
Necesidad	Alta	**Verificabilidad**	Alta
Prioridad	Alta	**Estabilidad**	Baja
Origen	CA_UR_06, CA_UR_07, CA_UR_09		

Tabla 42 RF_SR_14

RF_SR_15			
Descripción	El sistema permitirá copiar ficheros alojados fuera de los sistemas de ficheros montados a la estructura de carpetas accesible desde el punto de montaje.		
Necesidad	Media	**Verificabilidad**	Media
Prioridad	Baja	**Estabilidad**	Media
Origen	CA_UR_07		

Tabla 43 RF_SR_15

RF_SR_16			
Descripción	El sistema informará de los errores ocurridos al copiar ficheros a una carpeta accesible a través del punto de montaje.		
Necesidad	Media	**Verificabilidad**	Alta
Prioridad	Baja	**Estabilidad**	Media
Origen	CA_UR_07		

Tabla 44 RF_SR_16

RF_SR_17			
Descripción	El sistema podrá acceder al contenido de los ficheros alojados en los sistemas de ficheros montados.		
Necesidad	Alta	**Verificabilidad**	Media
Prioridad	Alta	**Estabilidad**	Alta
Origen	CA_UR_08		

Tabla 45 RF_SR_17

RF_SR_18			
Descripción	El sistema mostrará los errores ocurridos al acceder al contenido de los ficheros alojados en los sistemas de ficheros montados.		
Necesidad	Media	**Verificabilidad**	Alta
Prioridad	Media	**Estabilidad**	Media
Origen	CA_UR_08		

Tabla 46 RF_SR_18

RF_SR_19			
Descripción	El sistema permitirá renombrar ficheros alojados dentro de los sistemas de ficheros montados.		
Necesidad	Media	**Verificabilidad**	Alta
Prioridad	Media	**Estabilidad**	Media
Origen	CA_UR_09		

Tabla 47 RF_SR_19

RF_SR_20			
Descripción	El sistema mostrará los errores ocurridos al renombrar ficheros alojados dentro de los sistemas de ficheros montados.		
Necesidad	Media	**Verificabilidad**	Alta
Prioridad	Media	**Estabilidad**	Media
Origen	CA_UR_09		

Tabla 48 RF_SR_20

RF_SR_21			
Descripción	El sistema permitirá montar los sistemas de ficheros en el punto de montaje a través de la línea de comandos		
Necesidad	Alta	**Verificabilidad**	Alta
Prioridad	Alta	**Estabilidad**	Media
Origen	CA_UR_10		

Tabla 49 RF_SR_21

3.3.2 Requisitos no funcionales

RNF_SR_01			
Descripción	El sistema mostrará los errores ocurridos al montar los diferentes sistemas de ficheros.		
Necesidad	Alta	**Verificabilidad**	Media
Prioridad	Alta	**Estabilidad**	Media
Origen	RE_UR_01		

Tabla 50 RNF_SR_01

RNF_SR_02			
Descripción	El sistema mostrará, en caso de error en los parámetros de montaje, el formato correcto necesario para montar los sistemas de ficheros		
Necesidad	Media	**Verificabilidad**	Alta
Prioridad	Media	**Estabilidad**	Alta

RNF_SR_02	
Origen	RE_UR_01

Tabla 51 RNF_SR_02

RNF_SR_03			
Descripción	El sistema se implementará en el lenguaje de programación c.		
Necesidad	Media	**Verificabilidad**	Alta
Prioridad	Alta	**Estabilidad**	Alta
Origen	RE_UR_02, RE_UR_05		

Tabla 52 RNF_SR_03

RNF_SR_04			
Descripción	El sistema mantendrá un fichero en el que guardará los sucesos relacionados con las operaciones soportadas.		
Necesidad	Baja	**Verificabilidad**	Media
Prioridad	Baja	**Estabilidad**	Alta
Origen	RE_UR_03		

Tabla 53 RNF_SR_04

RNF_SR_05			
Descripción	El sistema se implementará utilizando la librería FUSE en su versión 2.9.3 para la gestión de las operaciones realizadas sobre los sistemas de ficheros		
Necesidad	Alta	**Verificabilidad**	Alta
Prioridad	Alta	**Estabilidad**	Alta

RNF_SR_05	
Origen	RE_UR_02

Tabla 54 RNF_SR_05

RNF_SR_06			
Descripción	El sistema se ejecuta sobre el kernel de linux 3.13 tanto en su versión de 32 bits como en la de 64 bits		
Necesidad	Alta	**Verificabilidad**	Alta
Prioridad	Alta	**Estabilidad**	Alta
Origen	RE_UR_05		

Tabla 55 RNF_SR_06

RNF_SR_07			
Descripción	El sistema será compatible con los siguientes sistemas de ficheros: • EXT 2 • EXT 3 • EXT 4 • XFS • JFS		
Necesidad	Alta	**Verificabilidad**	Baja
Prioridad	Alta	**Estabilidad**	Alta
Origen	RE_UR_05		

Tabla 56 RNF_SR_07

RNF_SR_08			
Descripción	El sistema creará los ficheros y carpetas con permisos de escritura, lectura y ejecución para todos los usuarios		
Necesidad	Media	**Verificabilidad**	Alta
Prioridad	Alta	**Estabilidad**	Alta
Origen	RE_UR_04		

Tabla 57 RNF_SR_08

3.4 Matriz de Trazabilidad

A continuación, se muestran las matrices de trazabilidad. Esta herramienta permite, como su nombre indica, trazar el origen de cada uno de los requisitos o funcionalidades y la evolución sufrida desde la fase de diseño hasta ser implementada.

3.4.1 Trazabilidad SR-UR

En esta sección, se muestra la trazabilidad entre los requisitos de software y los requisitos de usuario

3.4.1.1 Requisitos funcionales / Requisitos de usuario (Capacidad)

	CA_UR_01	CA_UR_02	CA_UR_03	CA_UR_04	CA_UR_05	CA_UR_06	CA_UR_07	CA_UR_08	CA_UR_09	CA_UR_10	CA_UR_11
RF_SR_01										X	X
RF_SR_02											X
RF_SR_03	X							X			
RF_SR_04	X							X			
RF_SR_05		X									
RF_SR_06		X									

	CA_UR_01	CA_UR_02	CA_UR_03	CA_UR_04	CA_UR_05	CA_UR_06	CA_UR_07	CA_UR_08	CA_UR_09	CA_UR_10	CA_UR_11
RF_SR_07			X								
RF_SR_08			X								
RF_SR_09				X							
RF_SR_10				X							
RF_SR_11					X						
RF_SR_12						X					
RF_SR_13						X					
RF_SR_14						X	X		X		
RF_SR_15							X				
RF_SR_16							X				
RF_SR_17							X				
RF_SR_18								X			
RF_SR_19									X		
RF_SR_20									X		
RF_SR_21										X	
RF_SR_22											X

Tabla 58 Trazabilidad RF/RU

3.4.1.2 Requisitos no funcionales / Requisitos de usuario (Restricción)

	RE_UR_01	RE_UR_02	RE_UR_03	RE_UR_04	RE_UR_05	RE_UR_06	RE_UR_07	RE_UR_08
RNF_SR_01	X	X						
RNF_SR_02			X		X			
RNF_SR_03				X				
RNF_SR_04								X
RNF_SR_05			X			X	X	

Tabla 59 Trazabilidad RNF/RU

4 Diseño de la solución

4.1 Descripción de la solución

Para alcanzar los objetivos marcados por los requisitos del sistema, será necesario usar *FUSE* como puente entre el espacio del usuario y las operaciones a nivel de *kernel*. De esta forma, el sistema implementará una serie de funciones que *FUSE* utilizará en lugar de las proporcionadas por el *kernel*.

Estudiando los casos de usos descritos en la sección de análisis, obtenemos la siguiente lista de funciones de *kernel* que deben ser implementadas:

Acción realizada	Funciones de kernel untilizadas
Mkdir	mkdir() getattr()
Rmdir	rmdir() getattr()
Ls	getattr()
Cd	getattr()
Touch	create() utime() getattr()
Cat	open() read() getattr()
Mv	rename() getattr()
Rm	unlink() getattr()

Tabla 60 Funciones por operaciones

Uno de los puntos críticos a tener en cuenta a la hora de realizar el diseño de la solución, consiste en determinar una política de asignación de sistemas de ficheros en la creación y copiado de archivos. En esta primera implementación, se opta por utilizar una política *Round Robin* de asignación. Aplicando esta política, cada uno de los nuevos ficheros creados o copiados, será almacenado en un sistema de ficheros diferente y en un orden determinado por la aplicación.

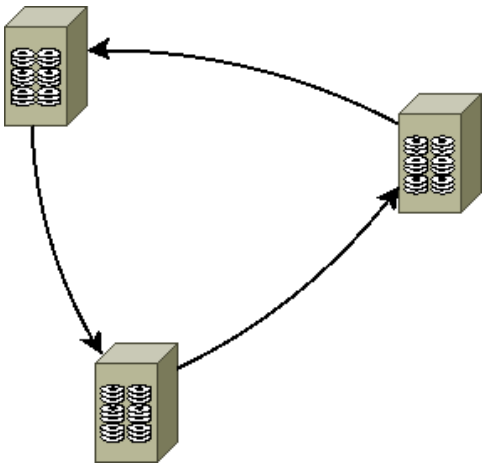


Ilustración 9 Esquema Round Robin

Un problema similar surge en la creación de nuevas carpetas. Al igual que en el caso de los archivos, es necesario establecer un criterio de creación de carpetas en los sistemas de ficheros montados. En este caso, por simplicidad y para mantener la coherencia entre los distintos sistemas montados, se opta por la creación de la misma estructura de carpetas en todos los sistemas de ficheros montados. De esta forma, al crear una carpeta a través del punto de montaje, está será creada en todos los sistemas.

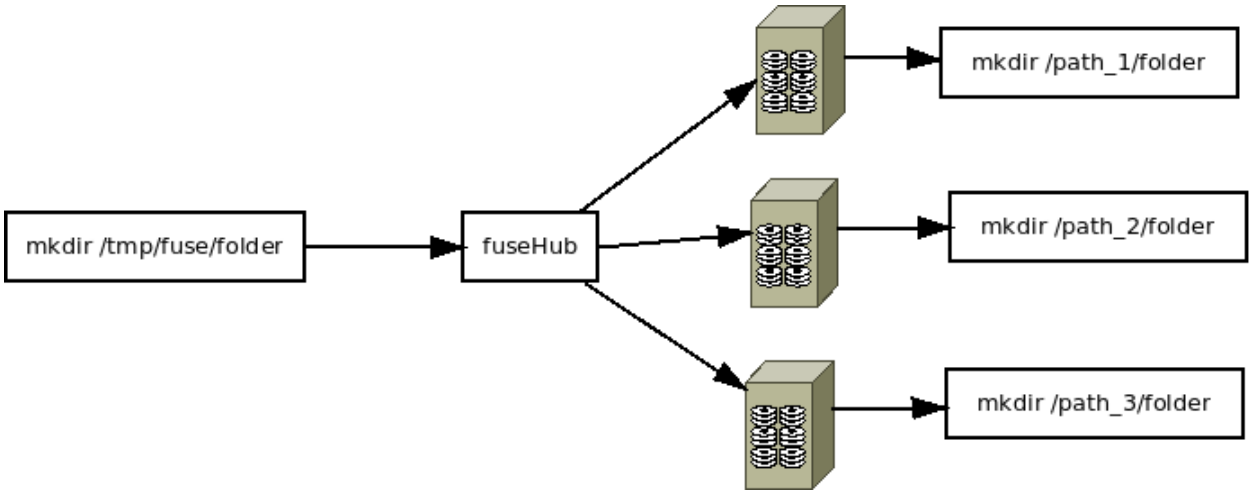


Ilustración 10 Esquema de funcionamiento de la aplicación

Este tipo de redundancia simplificará las comprobaciones necesarias a la hora del copiado o creación de ficheros. En contraposición, el proceso de borrado de carpetas se vuelve más complejo al tener que eliminar, como resulta evidente, la carpeta de todos los sistemas.

4.2 Funciones implementadas

En esta sección, se describen las funciones del sistema que sustituirán, a través de *FUSE*, a las del sistema operativo para poder acceder a las estructuras de datos de diferentes sistemas de ficheros.

Función:	***f_getattr***
Sustituye a:	***getattr***
Descripción:	***Obtiene los atributos de un fichero***. Para conseguirlo, localiza el sistema de ficheros en el que se encuentra y obtiene sus atributos

Tabla 61 getattr

Función:	***f_readdir***
Sustituye a:	***readdir***
Descripción:	***Lee un directorio.*** En la implementación del sistema, se localizan la ruta en todos los sistemas de ficheros, se lee el contenido de todas las rutas localizadas y se introducen los resultados en el buffer de contenidos.

Tabla 62 readdir

Función:	***f_open***
Sustituye a:	***open***
Descripción:	***Abre un fichero***. La primera operación necesaria es la de localizar el sistemas de ficheros en el que se aloja el fichero. Una vez localizado, se obtiene su id y se inserta dentro fi, estructura de datos proporcionada por FUSE para localizar

	ficheros. Tras esta operación, se incluye en una estructura de datos con los ficheros abiertos.

Tabla 63 open

Función:	***f_read***
Sustituye a:	***read***
Descripción:	***Lee los datos de un fichero abierto***. El primer paso implementado consiste en obtener el identificador del fichero abierto que se almaceno en fi. Una vez obtenido, se busca en la estructura de ficheros abiertos y se leen sus datos.

Tabla 64 read

Función:	***f_create***
Sustituye a:	***create***
Descripción:	***Crea un fichero abierto.*** La primera fase consiste en la localización del sistema de ficheros en el que se va a almacenar el fichero creado. En la implementación actual se ha optado por una política Round Robin. Una vez seleccionado el sistema de ficheros, se crea el fichero en la ruta adecuada.

Tabla 65 create

Función:	***f_mkdir***
Sustituye a:	***mkdir***
Descripción:	***Crea un directorio.*** Para realizar esta operación, se creará un directorio con la ruta recibida por parámetro en todos los sistemas de ficheros montados.

Tabla 66 mkdir

Función:	***f_rmdir***

Sustituye a:	***rmdir***
Descripción:	***Borra un directorio.*** En esta función, se itera por todos los sistemas de ficheros montados y se elimina la carpeta recibida por parámetro.

Tabla 67 rmdir

Función:	***f_unlink***
Sustituye a:	***unlink***
Descripción:	**Elimina un fichero.** El primer paso consiste en localizar el sistema de ficheros en el que se encuentra el archivo. Una vez localizada, se borra.

Tabla 68 unlink

Función:	***f_rename***
Sustituye a:	***rename***
Descripción:	***Renombra un fichero.*** Una vez localizado el archivo, se renombra, manteniéndolo en el mismo sistema de ficheros.

Tabla 69 rename

Función:	***f_write***
Sustituye a:	***write***
Descripción:	***Escribe datos a un fichero abierto.*** Obtiene la id del archivo sobre el que escribir a través de FUSE y vuelca los datos en él, manteniéndolo dentro del mismo sistema de ficheros.

Tabla 70 write

4.3 Diagrama de estructura de datos

A continuación, se muestra el modelo UML con las estructuras de datos utilizadas para satisfacer las necesidades funcionales del sistema. Cada uno de los elementos será descrito posteriormente.

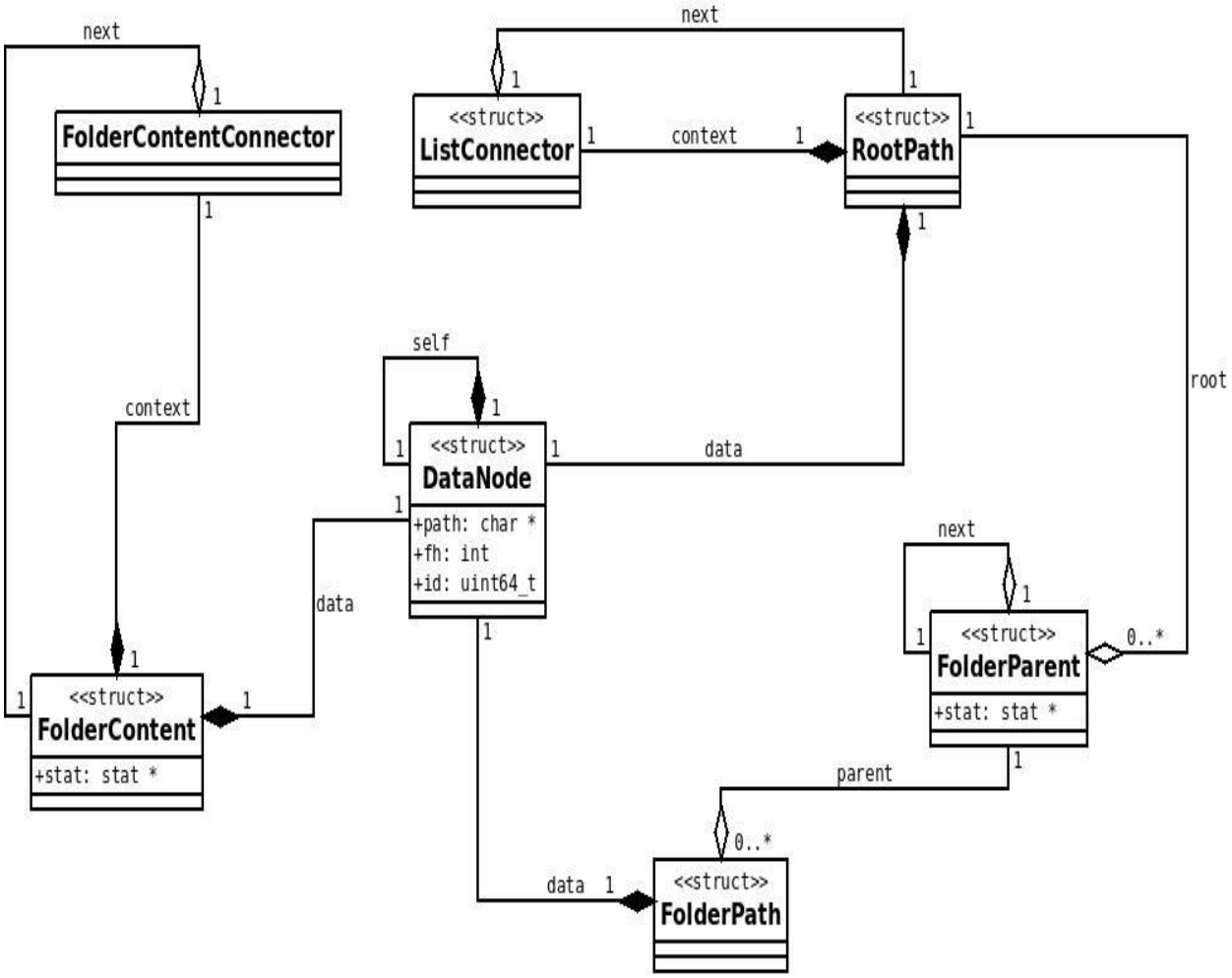


Ilustración 11 Diagrama de estructura de datos

4.3.1 Almacenamiento de rutas de sistemas de ficheros.

El sistema necesita una estructura de datos para almacenar las rutas de los sistemas de ficheros montados. Para ello, se utilizarán las siguientes estructuras de datos:

- ***RootPath***: Representa a cada uno de los sistemas de ficheros montados. Consta de las siguientes variables:
    - **context**: Puntero a *ListConnector*. Con esta relación, se crea una lista enlazada con todos los sistemas de ficheros montados.

- **data**: Puntero a *DataNode*. Almacenará la ruta del sistema de ficheros.

- ***DataNode***: Almacena la información relacionada con una ruta en el sistema. Consta de las siguientes variables:

  - **path**: Cadena de texto con la ruta del sistema de ficheros.

  - **fh**: Identificador utilizado por *FUSE* para el manejo de ficheros.

  - **self**: Puntero a *DataNode*. Almacena la dirección de memoria de la estructura de datos.

  - **id**: *Id* del objeto obtenida de la propiedad *self* a través de una unión entre estas dos propiedades.

- ***ListConnector***: Mediante esta estructura de datos, definiremos una lista enlazada en la que se almacenarán todas las rutas montadas. Está compuesta por:

  - ***next***: Puntero a *RootPath*. Representa al siguiente elemento de la lista.

4.3.2 Almacenamiento de rutas y contenido.

En ciertas operaciones implementadas, es necesario mantener un registro de rutas visitadas así como de su contenido. Para ello, se utilizarán las siguientes estructuras de datos:

- ***FolderPath***: Almacena la ruta de un ruta accedida a través del punto de montaje. Consta de las siguientes variables:

  - **data**: Puntero a *DataNode*. Almacenará la ruta además de otra información útil (Descrito en la sección anterior).

  - **parent**: Puntero a *FolderParent*: Almacena, en una lista enlazada, una relación con todos los sistemas de ficheros en los que se puede acceder a la ruta y el estado de los mismos.

- ***FolderParent***: Lista enlazada en la que se almacena el estado de una ruta con respecto a los sistemas de ficheros montados. Está formada por:

  - **root**: Puntero a *RootPath*. Almacena una referencia al punto de montaje al que es relativo el estado.

  - **stat**: Estado de la ruta en el sistema de ficheros.

  - **next**: Puntero a *FolderParent*. Representa al estado en el siguiente sistema de ficheros.

- ***FolderContent***: Esta estructura almacena un elemento correspondiente al contenido de una ruta. Su estructura es la siguientes:

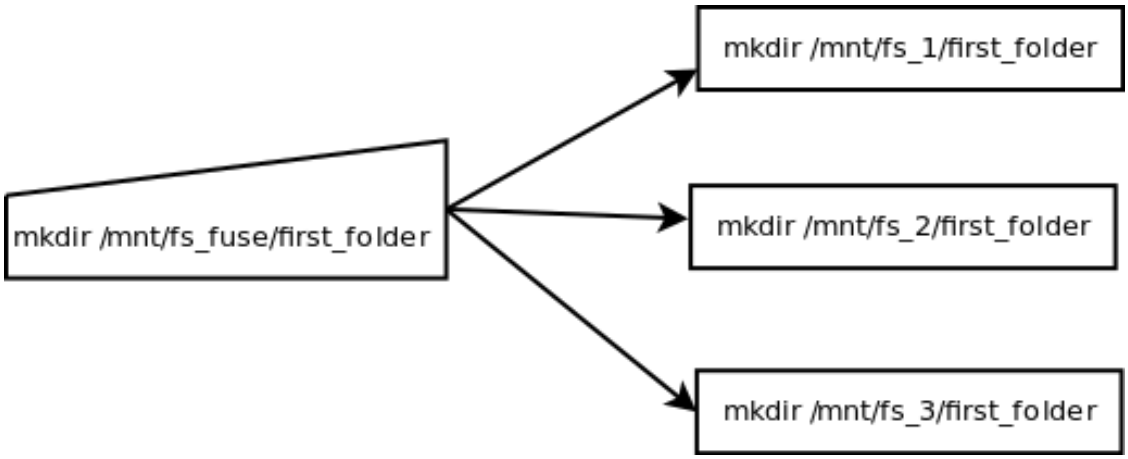
- **data**: Puntero a *DataNode*. Almacena la ruta del contenido con respecto al punto de montaje.
- **stat**: Estado del contenido en el sistema de ficheros que esté almacenado.
- **context**: Puntero a *FolderContentConnector*. Referencia a la lista enlazada con el resto de los contenidos.

- **FolderContentConnector**: En esta estructura de datos se definirá una lista enlazada en la que se almacenará todo el contenido. Consta de:
  - ***next***: Puntero a *FolderContent*. Representa al siguiente elemento con contenido.

4.4 Política de asignación y localización de espacios

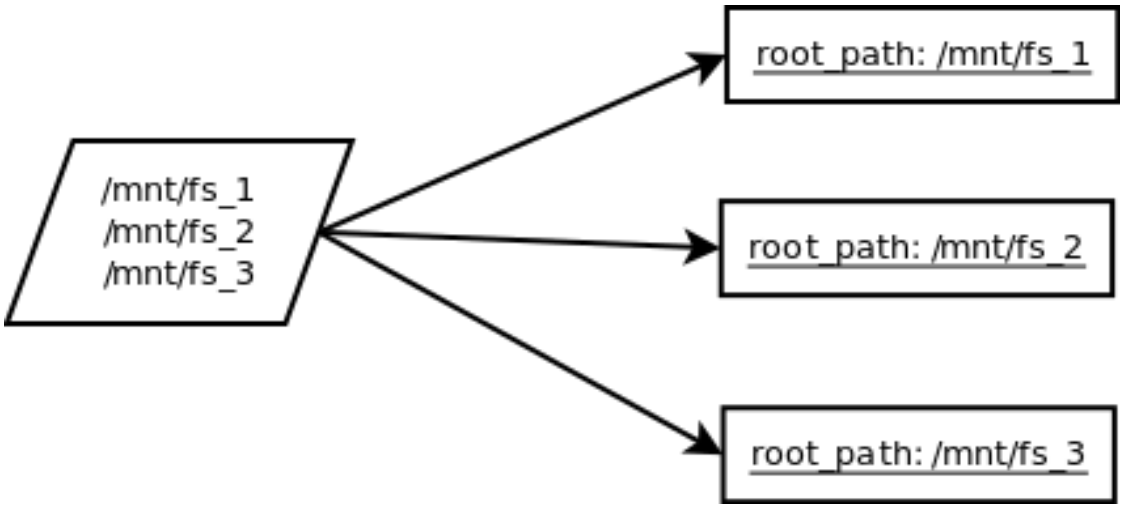
Como se ha descrito en secciones anteriores, la operación de creado de carpetas se gestiona replicando la misma estructura carpetas en todos los sistemas de ficheros montados.

La creación redundante de carpetas en los sistemas de ficheros, permite una gestión más sencilla de las operaciones de creación y copiado. Esto se justifica en ambos casos, ya que al realizar una operación de creación sobre una ruta creada anteriormente, siguiendo la política implementada, no resulta necesario realizar comprobaciones que indiquen si la ruta existe en el sistema de ficheros en el que se va a escribir, ni realizar operaciones de creación de carpetas a la hora de crear o copiar un fichero.



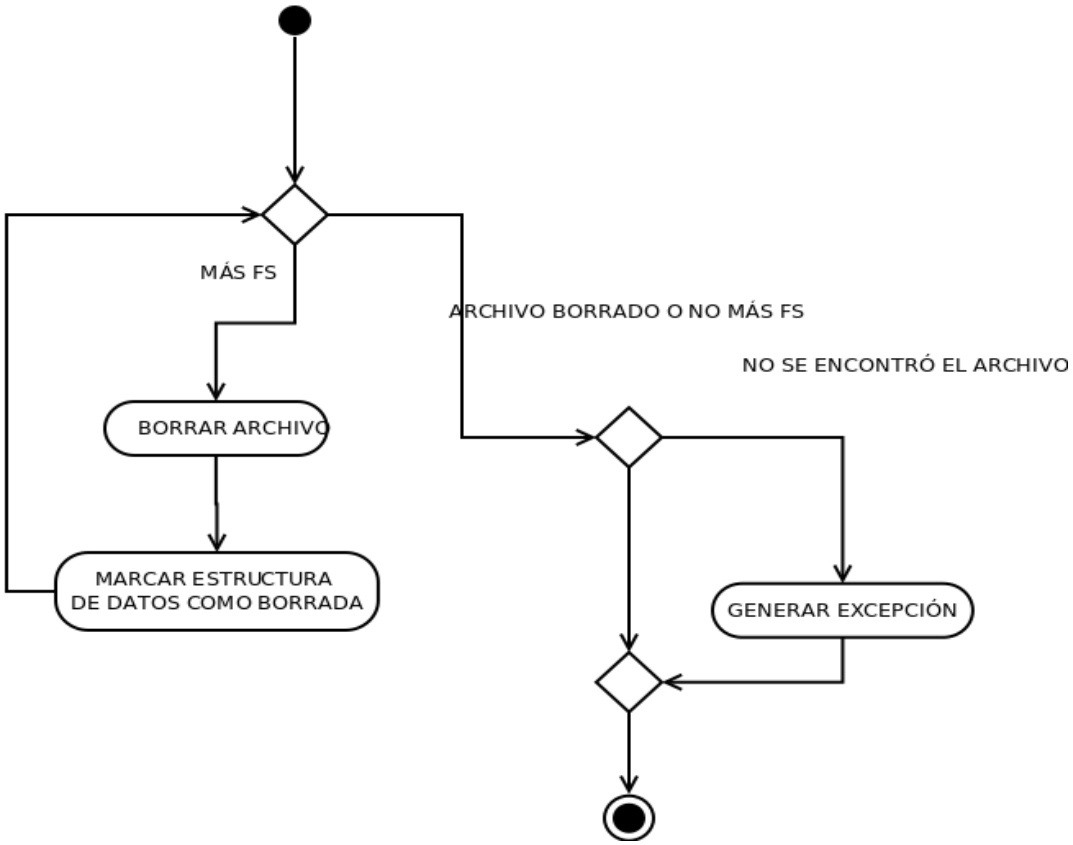
En el caso de la creación y copiado de ficheros, el sistema localizará en primer lugar, el sistema de ficheros en el que se deben escribir los datos. Actualmente se utiliza un sistema Round Robin para determinar la ruta que debe utilizarse. Para ello, en el arranque de la aplicación, debe indicarse mediante un parámetro, los sistemas de ficheros que van a ser

montados. El sistema los almacenara en una lista enlazada de datos en la que cada una de las rutas estará representada mediante una estructura *RootPath*.

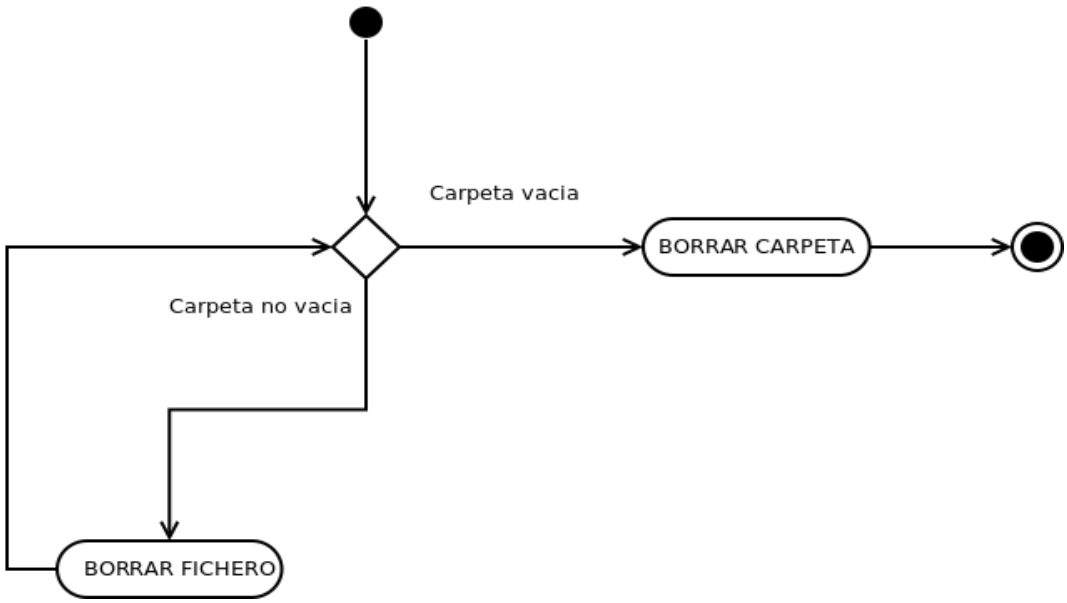


Una vez cargadas las rutas, el sistema determinará para cada operación la ruta asignada iterando sobre la lista y volviendo al comienzo al llegar a su final.

Para el borrado de ficheros, se localizara la ruta que se desea borrar iterando sobre la lista enlazada con las rutas montadas hasta encontrar el fichero buscado. En caso de no encontrarse, se producirá un error.



El caso más complejo se produce al borrar una carpeta con contenido. En este caso se debe iterar sobre todos los sistemas de ficheros montados y, en caso de que la ruta albergue contenido, borrarlo.



5 Planificación

5.1 Product Backlog

La planificación y estimación de un proyecto ágil depende en una sola métrica: La velocidad de desarrollo del equipo. Esta velocidad, calculada en función de proyectos anteriores, describe cuánto trabajo se puede realizar en cada una de las iteraciones y, aplicando dicha métrica, definir cuanto alcance puede conseguir el equipo para una fecha tope.

Tras una primera toma de requisitos, estos son analizados para obtener una lista de funcionalidades. Éstas son divididas en Objetos de la Pila de Producto *(Product Backlog Items),* a los que haremos referencia de ahora en adelante como *PBIs*, que se priorizan y sobre los que se realizan una estimación potencial en puntos de historia.

De esta forma, cada una de las funcionalidades solicitadas puede ser programada para ser entregada en una iteración concreta. El plan de entrega de estos *PBIs* es denominado Plan de Publicación.

En este proyecto vamos a utilizar iteraciones de cuatro semanas tras las cuales, y salvo en caso del Sprint 0, se realizará una demo mostrando los avances realizados *al Product Owner*. En ellas se mostrará el comportamiento del producto al enfrentarse a cada una de las situaciones descritas en el apartado “Criterios de aceptación” que se encuentra en la ficha del PBI.

A continuación se muestra el estado de la pila de producto tras el análisis de los requisitos funcionales y de usuario.

Para facilitar la lectura y trazabilidad, cada *PBI* será representado mediante una ficha que contará con los siguientes campos:

<table>
<tr><th>Código del PBI</th><th>Nombre</th></tr>
<tr><td>Tipo</td><td></td></tr>
<tr><td>Descripción</td><td></td></tr>
<tr><td>Criterio de aceptación</td><td>
<table>
<tr><th>Escenario N</th><th></th></tr>
<tr><td>Dado</td><td>• precondición 1
• precondición 2</td></tr>
<tr><td>Cuando</td><td>• acción</td></tr>
<tr><td>Entonces</td><td>• postcondición 1
• postcondición 2</td></tr>
<tr><td>Ejemplo</td><td>• Ejemplo 1
• Ejemplo 2</td></tr>
</table>
</td></tr>
<tr><td>Estimación</td><td></td></tr>
</table>

Tabla 71 Formato PBI

- **ID**: Identificador unívoco del PBI. Su nomenclatura seguirá el siguiente formato :
  - ***PBI_ZZ*** donde ZZ será sustituido por un numeral, comenzando por 01 e incrementado en uno en los siguientes *PBIs*.
- **Tipo**: Se van a distinguir entre dos tipos de *PBIs*:
  - Historia de usuario: Descripción de una funcionalidad que debe incorporar un sistema de software, y cuya implementación aporta valor al cliente.
  - Deuda técnica: Formación, análisis o prevención de posibles malas prácticas en el desarrollo del proyecto.
- **Nombre**: Nombre descriptivo del PBI. En el caso de Historias de usuario, este seguirá el siguiente formato: **Como "Tipo de usuario" quiero "acción a realizar" para "ganancia que se obtiene"**
- **Descripción**: Descripción detallada del requisito. Todos los detalles han de incluirse aquí para no dar lugar a ambigüedades.
- **Criterio de aceptación**: Se describirán casos específicos que deben ser probados para que la implementación del *PBI* se dé por aceptada en la fase de testing. Se seguirá la siguiente plantilla:

  Donde:

- **N** es un numeral que se incrementará en una unidad por cada escenario comenzando desde 01.
- **Precondición M**: Cada una de las precondiciones que deben darse para que se ejecute la acción.
- **Acción**: La acción que se va a realizar.
- **Postcondición L**: Cada una de las postcondiciones que se deben dar tras la acción.
- **Ejemplo**: Ejemplo de ejecución.
- **Estimación**: Estimación en puntos de historia del *PBI*.

5.1.1 Historias de usuario

<table>
<tr><td>PBI_01</td><td>Como usuario quiero compilar la aplicación utilizando la línea de comandos</td></tr>
<tr><td>Tipo</td><td>Historia de Usuario</td></tr>
<tr><td>Descripción</td><td>El usuario podrá compilar el sistema mediante el comando make.</td></tr>
<tr><td>Criterio de aceptación</td><td>
<table>
<tr><td>Escenario 1</td><td></td></tr>
<tr><td>Dado</td><td>El código fuente de la aplicación</td></tr>
<tr><td>Cuando</td><td>Se ejecuta el comando make desde la línea de comandos</td></tr>
<tr><td>Entonces</td><td>Se compilarán las distintas fuentes del producto y se generará un ejecutable</td></tr>
<tr><td>Ejemplo</td><td>$make</td></tr>
</table>
</td></tr>
<tr><td>Estimación</td><td>2</td></tr>
</table>

Tabla 72 PBI_01

PBI_02	**Como usuario, quiero montar los sistemas de ficheros en el punto de montaje a través de la línea de comandos para poder acceder a sus estructuras de archivos.**
Tipo	• Historia de Usuario

<table>
<tr><td>PBI_02</td><td>Como usuario, quiero montar los sistemas de ficheros en el punto de montaje a través de la línea de comandos para poder acceder a sus estructuras de archivos.</td></tr>
<tr><td>Descripción</td><td>El usuario puede ejecutar la aplicación desde la línea de comandos.
La aplicación recibirá como primer parámetro la ruta a un fichero de texto en el que se encontrarán almacenados las rutas que se desean montar en el hub.
El segundo parámetro será la ruta al punto de montaje del hub.</td></tr>
<tr><td>Dependencias</td><td>PBI_01</td></tr>
<tr><td>Criterio de aceptación</td><td><table>
<tr><td>Escenario 1</td><td></td></tr>
<tr><td>Dado</td><td>El ejecutable de la aplicación</td></tr>
<tr><td>Cuando</td><td>Ejecutamos la aplicación con los parámetros adecuados</td></tr>
<tr><td>Entonces</td><td>Se montan las rutas que se encuentran en el fichero que se pasa como primer parámetro en la ruta que se pasa como segundo parámetro.</td></tr>
<tr><td>Ejemplo</td><td>$./fusehub setup.conf /mnt/mounted</td></tr>
</table></td></tr>
<tr><td>Estimación</td><td>1</td></tr>
</table>

Tabla 73 PBI_02

PBI_03	**Como usuario, quiero recibir las instrucciones sobre los parámetros que debo pasarle a la aplicación para que se ejecute correctamente**
Tipo	• Historia de Usuario

<table>
<tr><th>PBI_03</th><td>Como usuario, quiero recibir las instrucciones sobre los parámetros que debo pasarle a la aplicación para que se ejecute correctamente</td></tr>
<tr><th>Descripción</th><td>En el caso en el que el usuario le pasa un número incorrecto de parámetros a la aplicación, se le mostrará un texto por consola indicando el número y formato correcto de los parámetros</td></tr>
<tr><th>Dependencias</th><td>PBI_02</td></tr>
<tr><th>Criterio de aceptación 1</th><td>
<table>
<tr><th>Escenario 1</th><th></th></tr>
<tr><th>Dado</th><td>El ejecutable de la aplicación</td></tr>
<tr><th>Cuando</th><td>Ejecutamos la aplicación con un número erróneo de parámetros</td></tr>
<tr><th>Entonces</th><td>Se muestra por consola un texto mostrando el número y formato de los parámetros</td></tr>
<tr><th>Ejemplo</th><td>$./fusehub setup.conf</td></tr>
</table>
</td></tr>
<tr><th>Criterio de aceptación 2</th><td>
<table>
<tr><th>Escenario 2</th><th></th></tr>
<tr><th>Dado</th><td>El ejecutable de la aplicación</td></tr>
<tr><th>Cuando</th><td>Ejecutamos la aplicación con un fichero que no existe como parámetro</td></tr>
<tr><th>Entonces</th><td>Se muestra por consola un texto mostrando el error</td></tr>
<tr><th>Ejemplo</th><td>$./fusehub setup.conf</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>PBI_03</th><th>Como usuario, quiero recibir las instrucciones sobre los parámetros que debo pasarle a la aplicación para que se ejecute correctamente</th></tr>
<tr><td>Criterio de aceptación 3</td><td>
<table>
<tr><th>Escenario 3</th><th></th></tr>
<tr><td>Dado</td><td>• El ejecutable de la aplicación</td></tr>
<tr><td>Cuando</td><td>• Ejecutamos la aplicación con un fichero con más sistemas de ficheros de los permitidos</td></tr>
<tr><td>Entonces</td><td>• Se muestra por consola el error</td></tr>
<tr><td>Ejemplo</td><td>• $./fusehub setup.conf</td></tr>
</table>
</td></tr>
<tr><td>Criterio de aceptación 4</td><td>
<table>
<tr><th>Escenario 4</th><th></th></tr>
<tr><td>Dado</td><td>• El ejecutable de la aplicación</td></tr>
<tr><td>Cuando</td><td>• Ejecutamos la aplicación con un fichero con menos sistemas de ficheros de los permitidos</td></tr>
<tr><td>Entonces</td><td>• Se muestra por consola el error</td></tr>
<tr><td>Ejemplo</td><td>• $./fusehub setup.conf</td></tr>
</table>
</td></tr>
<tr><td>Estimación</td><td>1</td></tr>
</table>

Tabla 74 PBI_03

<table>
<tr><th>PBI_04</th><th>Como usuario, quiero crear carpetas a través del punto de montaje</th></tr>
<tr><td>Tipo</td><td>Historia de Usuario</td></tr>
<tr><td>Descripción</td><td>El usuario puede crear carpetas dentro del punto de montaje del hub. Estas carpetas se replicarán en todas las carpetas root.</td></tr>
<tr><td>Dependencias</td><td>PBI_02</td></tr>
<tr><td>Criterio de aceptación</td><td>
<table>
<tr><th>Escenario 1</th><th></th></tr>
<tr><td>Dado</td><td>La aplicación se está ejecutando correctamente</td></tr>
<tr><td>Cuando</td><td>Se ejecuta el comando de creación de carpetas con una ruta que es accesible dentro del punto de montaje</td></tr>
<tr><td>Entonces</td><td>Se crea una carpeta en cada una de las carpetas root.
Se muestra sólo una carpeta dentro del hub.</td></tr>
<tr><td>Ejemplo</td><td>$ mkdir /mnt/mounted/foo</td></tr>
</table>
</td></tr>
<tr><td>Estimación</td><td>5</td></tr>
</table>

Tabla 75 PBI_04

PBI_05	Como usuario, quiero borrar carpetas de varios sistemas de ficheros a través del punto de montaje
Tipo	• Historia de Usuario

<table>
<tr><th>PBI_05</th><th>Como usuario, quiero borrar carpetas de varios sistemas de ficheros a través del punto de montaje</th></tr>
<tr><td>Descripción</td><td>La carpeta se borrará de todos los sistemas de ficheros montados a través del punto de montaje.</td></tr>
<tr><td>Dependencias</td><td>PBI_02</td></tr>
<tr><td>Criterio de aceptación 1</td><td>
<table>
<tr><th>Escenario 1</th><th></th></tr>
<tr><td>Dado</td><td>La aplicación se está ejecutando.
RUTA_1 es la ruta relativa al hub de una carpeta que existe en alguna de las carpetas root.</td></tr>
<tr><td>Cuando</td><td>Ejecutamos el comando de borrado de carpetas pasándole como parámetro RUTA_1.</td></tr>
<tr><td>Entonces</td><td>Se elimina la carpeta de todas las carpetas root.
No se muestra al listar el contenido de la carpeta.</td></tr>
<tr><td>Ejemplo</td><td>$ rmdir /mnt/mounted/RUTA_1</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>PBI_05</th><th>Como usuario, quiero borrar carpetas de varios sistemas de ficheros a través del punto de montaje</th></tr>
<tr><td>Criterio de aceptación 2</td><td>
<table>
<tr><th>Escenario 2</th><th></th></tr>
<tr><td>Dado</td><td>• La aplicación se está ejecutando.
• RUTA_2 es la ruta relativa punto de montaje.
• RUTA_2 no existe</td></tr>
<tr><td>Cuando</td><td>• Ejecutamos el comando de borrado de carpetas pasándole como parámetro RUTA_2.</td></tr>
<tr><td>Entonces</td><td>• Se muestra un mensaje de error indicando que la carpeta no existe</td></tr>
<tr><td>Ejemplo</td><td>• $ rmdir /mnt/mounted/RUTA_2</td></tr>
</table>
</td></tr>
<tr><td>Estimación</td><td>5</td></tr>
</table>

Tabla 76 PBI_05

PBI_06	Como usuario, no quiero que se borren carpetas de varios sistemas de ficheros a través del punto de montaje, cuando estas tienen contenido
Tipo	• Historia de Usuario
Descripción	• El usuario no podrá borrar carpetas a través del punto de montaje si estas tienen algún contenido.
Dependencias	PBI_02, PBI_05

<table>
<tr><th>PBI_06</th><th>Como usuario, no quiero que se borren carpetas de varios sistemas de ficheros a través del punto de montaje, cuando estas tienen contenido</th></tr>
<tr><td>Criterio de aceptación</td><td>
<table>
<tr><th>Escenario 1</th><th></th></tr>
<tr><td>Dado</td><td>• La aplicación se está ejecutando.
• RUTA_1 es la ruta relativa punto de montaje.
• RUTA_1 contiene algún fichero</td></tr>
<tr><td>Cuando</td><td>• Ejecutamos el comando de borrado de carpetas pasándole como parámetro RUTA_1.</td></tr>
<tr><td>Entonces</td><td>• Se muestra un mensaje de error indicando que la carpeta no está vacía</td></tr>
<tr><td>Ejemplo</td><td>• $ rmdir /mnt/mounted/RUTA_1</td></tr>
</table>
</td></tr>
<tr><td>Estimación</td><td>3</td></tr>
</table>

Tabla 77 PBI_06

PBI_07	Como usuario, quiero navegar por las carpetas de los sitemas de ficheros montados a través del punto de montaje
Tipo	• Historia de Usuario
Descripción	• El usuario podrá acceder a las carpetas que se encuentren en las carpetas *root* a través del *hub* de forma transparente.
Dependencias	PBI_02

<table>
<tr><th>PBI_07</th><th>Como usuario, quiero navegar por las carpetas de los sitemas de ficheros montados a través del punto de montaje</th></tr>
<tr><td>Criterio de aceptación 1</td><td>
<table>
<tr><th>Escenario 1</th><th></th></tr>
<tr><td>Dado</td><td>La aplicación se está ejecutando
RUTA_1 es la ruta relativa al punto de montaje de una carpeta que existe en alguna de los sistemas de ficheros montados</td></tr>
<tr><td>Cuando</td><td>Se ejecuta el comando de cambio de directorio pasándole como parámetro la RUTA_1.</td></tr>
<tr><td>Entonces</td><td>La ruta en la que se encuentra el usuario es relativa al punto de montaje</td></tr>
<tr><td>Ejemplo</td><td>$ cd /mnt/mounted/RUTA_1</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>PBI_07</th><th>Como usuario, quiero navegar por las carpetas de los sitemas de ficheros montados a través del punto de montaje</th></tr>
<tr><td>Criterio de aceptación 2</td><td>
<table>
<tr><th>Escenario 2</th><th></th></tr>
<tr><td>Dado</td><td>• La aplicación se está ejecutando
• RUTA_2 es la ruta relativa al punto de montaje de una carpeta que no existe</td></tr>
<tr><td>Cuando</td><td>• Se ejecuta el comando de cambio de directorio pasándole como parámetro la RUTA_2.</td></tr>
<tr><td>Entonces</td><td>• Se muestra un mensaje de error indicando que la carpeta no existe.</td></tr>
<tr><td>Ejemplo</td><td>• $ cd /mnt/mounted/RUTA_2</td></tr>
</table>
</td></tr>
<tr><td>Estimación</td><td>5</td></tr>
</table>

Tabla 78 PBI_07

PBI_08	Como usuario, quiero listar el contenido de una carpeta a través del punto de montaje
Tipo	• Historia de Usuario
Descripción	• El usuario listará el contenido de las carpetas almacenadas en las carpetas *root* a través del *hub* eliminando repetidos (algunas de las carpetas van a estar duplicadas en los diferentes *roots*).
Dependencias	PBI_02

<table>
<tr><th>PBI_08</th><th>Como usuario, quiero listar el contenido de una carpeta a través del punto de montaje</th></tr>
<tr><td>Criterio de aceptación 1</td><td>
<table>
<tr><th>Escenario 1</th><th></th></tr>
<tr><td>Dado</td><td>La aplicación se está ejecutando.
RUTA_1 es la ruta relativa al hub de una carpeta que existe en una de las carpetas root.</td></tr>
<tr><td>Cuando</td><td>Ejecutamos el comando de listado de contenidos pasándole como parámetro RUTA_1.</td></tr>
<tr><td>Entonces</td><td>Se muestra el contenido de la ruta relativa en todas las carpetas root.</td></tr>
<tr><td>Ejemplo</td><td>$ ls /mnt/mounted/RUTA_1</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>PBI_08</th><th>Como usuario, quiero listar el contenido de una carpeta a través del punto de montaje</th></tr>
<tr><td>Criterio de aceptación 2</td><td>
<table>
<tr><th>Escenario 2</th><th></th></tr>
<tr><td>Dado</td><td>• La aplicación se está ejecutando.
• RUTA_2 es la ruta relativa al punto de montaje
• RUTA_2 existe en varios de los sistemas de ficheros</td></tr>
<tr><td>Cuando</td><td>• Ejecutamos el comando de listado de contenidos pasándole como parámetro RUTA_2.</td></tr>
<tr><td>Entonces</td><td>• Se muestra el contenido de la ruta relativa en todas las carpetas root en las que existe RUTA_2.
• En el listado no se mostrarán los posibles duplicados</td></tr>
<tr><td>Ejemplo</td><td>• $ ls /mnt/mounted/RUTA_2</td></tr>
</table>
</td></tr>
<tr><td>Estimación</td><td>5</td></tr>
</table>

Tabla 79 PBI_08

PBI_09	Como usuario, quiero que se me indique, al realizar un listado del contenido de un directorio, si este no existe
Tipo	• Historia de Usuario
Descripción	• El usuario recibirá un mensaje de error si, al listar el contenido de un directorio cuya ruta es relativa al punto de montaje, este no existe en ninguno de los sistemas de ficheros montados.
Dependencias	PBI_02, PBI_08

<table>
<tr><th>PBI_09</th><th>Como usuario, quiero que se me indique, al realizar un listado del contenido de un directorio, si este no existe</th></tr>
<tr><td>Criterio de aceptación</td><td>
<table>
<tr><th>Escenario 1</th><th></th></tr>
<tr><td>Dado</td><td>• La aplicación se está ejecutando.
• RUTA_3 es la ruta relativa al punto de montaje
• RUTA_3 no existe</td></tr>
<tr><td>Cuando</td><td>• Ejecutamos el comando de listado de contenidos pasándole como parámetro RUTA_3.</td></tr>
<tr><td>Entonces</td><td>• Se muestra un mensaje de error indicando que la carpeta no existe</td></tr>
<tr><td>Ejemplo</td><td>• $ ls /mnt/mounted/RUTA_3</td></tr>
</table>
</td></tr>
<tr><td>Estimación</td><td>3</td></tr>
</table>

Tabla 80 PBI_09

PBI_10	Como usuario, quiero copiar archivos a uno de los sistemas de ficheros montados a través del punto de montaje
Tipo	• Historia de Usuario
Descripción	• El usuario podrá copiar un fichero a una carpeta que se encuentre en el punto de montaje. • El fichero se almacenará en una de las carpetas *root* de forma transparente para el usuario.
Dependencias	PBI_02

<table>
<tr><th>PBI_10</th><th>Como usuario, quiero copiar archivos a uno de los sistemas de ficheros montados a través del punto de montaje</th></tr>
<tr><td>Criterio de aceptación 1</td><td>
<table>
<tr><th>Escenario 1</th><th></th></tr>
<tr><td>Dado</td><td>La aplicación se está ejecutando.
RUTA_1 es la ruta relativa al hub.
RUTA_1 existe y es accesible a través del hub.
RUTA_2 es una ruta no relativa al hub
RUTA_2 corresponde a un fichero existente.</td></tr>
<tr><td>Cuando</td><td>Ejecutamos el comando de copia de ficheros pasando como primer parámetro RUTA_2 y como segundo parámetro RUTA_1.</td></tr>
<tr><td>Entonces</td><td>Se copia el fichero a una de las carpetas root.</td></tr>
<tr><td>Ejemplo</td><td>$ cp RUTA_2 /mnt/mounted/RUTA_1</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>PBI_10</th><th>Como usuario, quiero copiar archivos a uno de los sistemas de ficheros montados a través del punto de montaje</th></tr>
<tr><td>Criterio de aceptación 2</td><td>
<table>
<tr><th>Escenario 2</th><th></th></tr>
<tr><td>Dado</td><td>• La aplicación se está ejecutando.
• RUTA_3 es la ruta relativa al hub.
• RUTA_3 existe.</td></tr>
<tr><td>Cuando</td><td>• Se copian n + 1 ficheros a RUTA_3, siendo n el número de sistemas de ficheros montados</td></tr>
<tr><td>Entonces</td><td>• Los n primeros ficheros se copian a sistemas de ficheros diferentes y el último, en la misma carpeta que el primero</td></tr>
<tr><td>Ejemplo</td><td>• $ cp RUTA_2 /mnt/mounted/RUTA_1</td></tr>
</table>
</td></tr>
<tr><td>Estimación</td><td>8</td></tr>
</table>

Tabla 81 PBI_10

PBI_11	Como usuario, quiero recibir un mensaje de error si, al copiar un archivo al sistema de ficheros a través del punto de montaje, este no existe
Tipo	• Historia de Usuario
Descripción	• El usuario será informado si la ruta del fichero que pretende copiar no existe.
Dependencias	PBI_02

<table>
<tr><td>PBI_11</td><td>Como usuario, quiero recibir un mensaje de error si, al copiar un archivo al sistema de ficheros a través del punto de montaje, este no existe</td></tr>
<tr><td>Criterio de aceptación 1</td><td>
<table>
<tr><td>Escenario 1</td><td></td></tr>
<tr><td>Dado</td><td>• La aplicación se está ejecutando.
• RUTA_3 es la ruta relativa al hub.
• RUTA_3 no existe.
• RUTA_4 es una ruta no relativa al hub
• RUTA_4 corresponde a un fichero existente.</td></tr>
<tr><td>Cuando</td><td>• Ejecutamos el comando de copia de ficheros pasando como primer parámetro RUTA_4 y como segundo parámetro RUTA_3.</td></tr>
<tr><td>Entonces</td><td>• Se muestra un mensaje de error indicando que la ruta a la que se quiere copiar el fichero no existe.</td></tr>
<tr><td>Ejemplo</td><td>• $ cp RUTA_2 /mnt/mounted/RUTA_1</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>PBI_11</td><td colspan="2">Como usuario, quiero recibir un mensaje de error si, al copiar un archivo al sistema de ficheros a través del punto de montaje, este no existe</td></tr>
<tr><td>Criterio de aceptación 2</td><td colspan="2"><table>
<tr><td>Escenario 2</td><td></td></tr>
<tr><td>Dado</td><td>• La aplicación se está ejecutando.
• RUTA_5 es la ruta relativa al hub.
• RUTA_5 existe y es accesible a través del hub.
• RUTA_6 es una ruta no relativa al hub
• RUTA_6 corresponde a un fichero no existente.</td></tr>
<tr><td>Cuando</td><td>• Ejecutamos el comando de copia de ficheros pasando como primer parámetro RUTA_6 y como segundo parámetro RUTA_5.</td></tr>
<tr><td>Entonces</td><td>• Se muestra un mensaje de error indicando que el fichero que se desea copiar no existe</td></tr>
<tr><td>Ejemplo</td><td>• $ cp RUTA_6 /mnt/mounted/RUTA_5</td></tr>
</table></td></tr>
<tr><td>Estimación</td><td colspan="2">3</td></tr>
</table>

Tabla 82 PBI_11

PBI_12	**Como usuario quiero crear nuevos archivos en los sistemas de ficheros montados a través del punto de montaje**
Tipo	• Historia de Usuario

<table>
<tr><th>PBI_12</th><th>Como usuario quiero crear nuevos archivos en los sistemas de ficheros montados a través del punto de montaje</th></tr>
<tr><td>Descripción</td><td>El usuario podrá crear un fichero nuevo dentro de una carpeta accesible a través del hub.
El fichero será creado en una de las carpetas root siguiendo una política Round Robin.</td></tr>
<tr><td>Dependencias</td><td>PBI_02</td></tr>
<tr><td>Criterio de aceptación 1</td><td>
<table>
<tr><th>Escenario 1</th><th></th></tr>
<tr><td>Dado</td><td>La aplicación se está ejecutando.
RUTA_1 es la ruta relativa al hub.
RUTA_1 hace referencia a una estructura de carpetas existente</td></tr>
<tr><td>Cuando</td><td>Ejecutamos el comando para crear un nuevo fichero pasando como parámetro RUTA_1</td></tr>
<tr><td>Entonces</td><td>Se genera un fichero nuevo dentro de una de las carpetas root</td></tr>
<tr><td>Ejemplo</td><td>$ touch /mnt/mounted/foo/foo.txt</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>PBI_12</th><th>Como usuario quiero crear nuevos archivos en los sistemas de ficheros montados a través del punto de montaje</th></tr>
<tr><td>Criterio de aceptación 2</td><td>
<table>
<tr><th>Escenario 2</th><th></th></tr>
<tr><td>Dado</td><td>• La aplicación se está ejecutando.
• RUTA_2 es la ruta relativa al hub.
• RUTA_2 existe</td></tr>
<tr><td>Cuando</td><td>• Ejecutamos el comando para crear un nuevo fichero n + 1 veces en la carpeta RUTA_1, siendo n el número de sistemas de ficheros montados.</td></tr>
<tr><td>Entonces</td><td>• Los n primeros ficheros se crean en diferentes sistemas de ficheros y, el último en el mismo que el primero</td></tr>
<tr><td>Ejemplo</td><td>• $ touch /mnt/mounted/foo/foo.txt</td></tr>
</table>
</td></tr>
<tr><td>Estimación</td><td>5</td></tr>
</table>

Tabla 83 PBI_12

PBI_13	Como usuario, quiero recibir un mensaje de error si, al crear un archivo, este ya existe
Tipo	• Historia de Usuario
Descripción	• El usuario recibirá un mensaje de error si, al crear un fichero con una ruta relativa al punto de montaje, este ya existe.
Dependencias	PBI_02

<table>
<tr><th>PBI_13</th><th>Como usuario, quiero recibir un mensaje de error si, al crear un archivo, este ya existe</th></tr>
<tr><td>Criterio de aceptación 1</td><td>
<table>
<tr><th>Escenario 1</th><th></th></tr>
<tr><td>Dado</td><td>La aplicación se está ejecutando.
RUTA_2 es la ruta relativa al hub.
RUTA_2 hace referencia a una carpeta que no existe.</td></tr>
<tr><td>Cuando</td><td>Ejecutamos el comando para crear un nuevo fichero pasando como parámetro RUTA_1.</td></tr>
<tr><td>Entonces</td><td>Se muestra un mensaje de error indicando que la ruta en la que se quiere crear el fichero no existe.</td></tr>
<tr><td>Ejemplo</td><td>touch /mnt/mounted/foo.txt</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>PBI_13</th><th>Como usuario, quiero recibir un mensaje de error si, al crear un archivo, este ya existe</th></tr>
<tr><td>Criterio de aceptación 2</td><td>
<table>
<tr><th>Escenario 2</th><th></th></tr>
<tr><td>Dado</td><td>• La aplicación se está ejecutando.
• RUTA_3 es la ruta relativa al hub.
• RUTA_3 corresponde a un fichero que ya existe.</td></tr>
<tr><td>Cuando</td><td>• Ejecutamos el comando para crear un nuevo fichero pasando como parámetro RUTA_3.</td></tr>
<tr><td>Entonces</td><td>• El sistema no realiza ninguna acción de creación de ficheros en la ruta especificada.</td></tr>
<tr><td>Ejemplo</td><td>• $ touch /mnt/mounted/foo/foo.txt</td></tr>
</table>
</td></tr>
<tr><td>Estimación</td><td>3</td></tr>
</table>

Tabla 84 PBI_13

PBI_14	Como usuario, quiero abrir archivos alojados en los sistemas de ficheros montados a través del punto de montaje
Tipo	• Historia de Usuario
Descripción	• El usuario podrá abrir un fichero alojado en una de las carpetas *root* a través del punto de montaje del *hub*.
Dependencias	PBI_02

<table>
<tr><td>PBI_14</td><td>Como usuario, quiero abrir archivos alojados en los sistemas de ficheros montados a través del punto de montaje</td></tr>
<tr><td>Criterio de aceptación 1</td><td>
<table>
<tr><td>Escenario 1</td><td></td></tr>
<tr><td>Dado</td><td>• La aplicación se está ejecutando.
• RUTA_1 es la ruta relativa al hub.
• RUTA_1 hace referencia a un fichero existente.</td></tr>
<tr><td>Cuando</td><td>• Ejecutamos un comando para obtener el contenido de un fichero pasando como parámetro RUTA_1.</td></tr>
<tr><td>Entonces</td><td>• Se muestra el contenido del fichero.</td></tr>
<tr><td>Ejemplo</td><td>• $ cat /mnt/mounted/foo/foo.txt</td></tr>
</table>
</td></tr>
<tr><td>Criterio de aceptación 2</td><td>
<table>
<tr><td>Escenario 2</td><td></td></tr>
<tr><td>Dado</td><td>• La aplicación se está ejecutando.
• RUTA_2 es la ruta relativa al hub.
• RUTA_2 hace referencia a un fichero que no existente.</td></tr>
<tr><td>Cuando</td><td>• Ejecutamos un comando para obtener el contenido de un fichero pasando como parámetro RUTA_2.</td></tr>
<tr><td>Entonces</td><td>• Se muestra un mensaje indicando que el fichero no existe</td></tr>
<tr><td>Ejemplo</td><td>• $ cat /mnt/mounted/foo/foo.txt</td></tr>
</table>
</td></tr>
<tr><td>Estimación</td><td>5</td></tr>
</table>

Tabla 85 PBI_14

<table>
<tr><th>PBI_15</th><th>Como usuario quiero borrar archivos alojados en los sistemas de ficheros montados a través del punto de montaje</th></tr>
<tr><td>Tipo</td><td>Historia de Usuario</td></tr>
<tr><td>Descripción</td><td>El usuario podrá borrar ficheros alojados en las carpetas root a través del punto de montaje.
El sistema localizará la carpeta root en la que se encuentra el fichero y lo eliminará.</td></tr>
<tr><td>Dependencias</td><td>PBI_02</td></tr>
<tr><td>Criterio de aceptación 1</td><td>
<table>
<tr><th>Escenario 1</th><th></th></tr>
<tr><td>Dado</td><td>La aplicación se está ejecutando.
RUTA_1 es la ruta relativa al hub.
RUTA_1 hace referencia a un fichero existente.</td></tr>
<tr><td>Cuando</td><td>Ejecutamos el comando de borrado de ficheros pasando como parámetro RUTA_1.</td></tr>
<tr><td>Entonces</td><td>Se elimina el fichero de la carpeta root en la que se encuentre.</td></tr>
<tr><td>Ejemplo</td><td>$ rm /mnt/mounted/foo/foo.txt</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>PBI_15</th><th>Como usuario quiero borrar archivos alojados en los sistemas de ficheros montados a través del punto de montaje</th></tr>
<tr><td>Criterio de aceptación 2</td><td>
<table>
<tr><th>Escenario 2</th><th></th></tr>
<tr><td>Dado</td><td>• La aplicación se está ejecutando.
• RUTA_2 es la ruta relativa al hub.
• RUTA_2 hace referencia a un fichero que no existente.</td></tr>
<tr><td>Cuando</td><td>• Ejecutamos el comando de borrado de ficheros pasando como parámetro RUTA_2.</td></tr>
<tr><td>Entonces</td><td>• Se muestra un mensaje de error indicando que el fichero no existe.</td></tr>
<tr><td>Ejemplo</td><td>• $ rm /mnt/mounted/foo/foo.txt</td></tr>
</table>
</td></tr>
<tr><td>Estimación</td><td></td></tr>
</table>

Tabla 86 PBI_15

PBI_16	Como usuario quiero renombrar archivos y carpetas alojados en los sistemas de ficheros montados a través del punto de montaje.
Tipo	• Historia de Usuario
Descripción	• El usuario podrá renombrar un fichero alojado en una de las carpetas *root* a través del punto de montaje.
Dependencias	PBI_02

<table>
<tr><td>PBI_16</td><td>Como usuario quiero renombrar archivos y carpetas alojados en los sistemas de ficheros montados a través del punto de montaje.</td></tr>
<tr><td>Criterio de aceptación 1</td><td>
<table>
<tr><td>Escenario 1</td><td></td></tr>
<tr><td>Dado</td><td>• La aplicación se está ejecutando.
• RUTA_1 es la ruta relativa al hub.
• RUTA_1 hace referencia a un fichero existente.
• RUTA_2 es la ruta relativa al hub.
• RUTA_2 corresponde a un fichero que no existe.
• RUTA_2 se encuentra en la misma carpeta que RUTA_1</td></tr>
<tr><td>Cuando</td><td>• Ejecutamos el comando de renombrado de ficheros pasando como parámetros RUTA_1 y RUTA_2.</td></tr>
<tr><td>Entonces</td><td>• Se renombra el fichero.
• Se mantiene en la misma carpeta root en la que se encuentre.</td></tr>
<tr><td>Ejemplo</td><td>• $ mv /mnt/mounted/foo/foo.txt /mnt/mounted/foo/other_name.txt</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>PBI_16</td><td>Como usuario quiero renombrar archivos y carpetas alojados en los sistemas de ficheros montados a través del punto de montaje.</td></tr>
<tr><td>Criterio de aceptación 2</td><td>
<table>
<tr><td>Escenario 2</td><td></td></tr>
<tr><td>Dado</td><td>• La aplicación se está ejecutando.
• RUTA_3 es la ruta relativa al hub.
• RUTA_3 hace referencia a un fichero existente.
• RUTA_4 es la ruta relativa al hub.
• RUTA_4 corresponde a un fichero existente.
• RUTA_4 se encuentra en la misma carpeta que RUTA_3</td></tr>
<tr><td>Cuando</td><td>• Ejecutamos el comando de renombrado de ficheros pasando como parámetros RUTA_3 y RUTA_4.</td></tr>
<tr><td>Entonces</td><td>• Se muestra un mensaje de error indicando que ya existe un fichero con ese nombre</td></tr>
<tr><td>Ejemplo</td><td>• $ mv /mnt/mounted/foo/foo.txt /mnt/mounted/foo/other_name.txt</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>PBI_16</td><td colspan="2">Como usuario quiero renombrar archivos y carpetas alojados en los sistemas de ficheros montados a través del punto de montaje.</td></tr>
<tr><td>Criterio de aceptación 3</td><td colspan="2"><table>
<tr><td>Escenario 3</td><td></td></tr>
<tr><td>Dado</td><td>• La aplicación se está ejecutando.
• RUTA_5 es la ruta relativa al hub.
• RUTA_5 hace referencia a un fichero que no existe.
• RUTA_6 es la ruta relativa al hub.
• RUTA_6 corresponde a un fichero existente.
• RUTA_6 se encuentra en la misma carpeta que RUTA_5.</td></tr>
<tr><td>Cuando</td><td>• Ejecutamos el comando de renombrado de ficheros pasando como parámetros RUTA_5 y RUTA_6.</td></tr>
<tr><td>Entonces</td><td>• Se muestra un mensaje de error indicando que el fichero que se pretende renombrar no existe.</td></tr>
<tr><td>Ejemplo</td><td>• $ mv /mnt/mounted/foo/foo.txt /mnt/mounted/foo/other_name.txt</td></tr>
</table></td></tr>
<tr><td>Estimación</td><td colspan="2">5</td></tr>
</table>

Tabla 87 PBI_16

5.1.2 Tareas Técnicas

PBI_17	Investigación Sobre la tecnología FUSE
Tipo	• Tarea técnica
Descripción	• Búsqueda de documentación, estudio y análisis de las peculiaridades del sistema.
Estimación	5

Tabla 88 PBI_17

PBI_18	Prueba de concepto de un sistema FUSE
Tipo	• Tarea técnica
Descripción	• Creación de una herramienta simple basada en FUSE que nos sirva para comprender mejor el funcionamiento de la librería y que utilizaremos como base para la implementación del módulo.
Estimación	5

Tabla 89 PBI_18

5.2 Matriz de trazabilidad

	PBI_01	PBI_02	PBI_03	PBI_04	PBI_05	PBI_06	PBI_07	PBI_08	PBI_09	PBI_10	PBI_11	PBI_12	PBI_13	PBI_14	PBI_15	PBI_16
RF_SR_01		x														
RF_SR_02			x													
RF_SR_03												x	x	x	x	x
RF_SR_04								x	x			x	x	x	x	x
RF_SR_05				x												
RF_SR_06				x												
RF_SR_07							x									
RF_SR_08							x									
RF_SR_09					x	x										
RF_SR_10					x	x										
RF_SR_11								x	x							
RF_SR_12												x	x			
RF_SR_13												x	x			
RF_SR_14										x	x	x	x			x

	PBI_01	PBI_02	PBI_03	PBI_04	PBI_05	PBI_06	PBI_07	PBI_08	PBI_09	PBI_10	PBI_11	PBI_12	PBI_13	PBI_14	PBI_15	PBI_16
RF_SR_15										x	x					
RF_SR_16										x	x					
RF_SR_17														x	x	
RF_SR_18														x	x	
RF_SR_19																x
RF_SR_20																x
RF_SR_21		x														
RNF_SR_01		x														
RNF_SR_02			x													
RNF_SR_03	x															
RNF_SR_04	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
RNF_SR_05	x															
RNF_SR_06	x															

	PBI_01	PBI_02	PBI_03	PBI_04	PBI_05	PBI_06	PBI_07	PBI_08	PBI_09	PBI_10	PBI_11	PBI_12	PBI_13	PBI_14	PBI_15	PBI_16
RNF_SR_07	x															
RNF_SR_08				x						x	x	x	x			

Tabla 90 Trazabilidad PBI/RF

5.3 Plan de Publicación

Tras identificar, detallar y estimar cada uno de los *PBIs*, se realiza una reunión con el *Product Owner* en la que se prioriza la pila de producto.

Antes de la tarea de priorización y para que el *Product Owner* tenga una imagen clara de cuál es el alcance de cada uno de los ítems, es necesario que el equipo exponga, con claridad que es lo que abarca cada PBI. Con esta información, el *Product Owner* tiene toda la información necesaria para establecer un orden dentro de la Pila de producto.

Una vez ordenada, y conociendo la velocidad de desarrollo del equipo, se puede establecer un plan de publicación de versiones. En nuestro caso, sabiendo que el equipo es capaz de desarrollar once puntos de historia por sprint, se obtiene el siguiente plan:

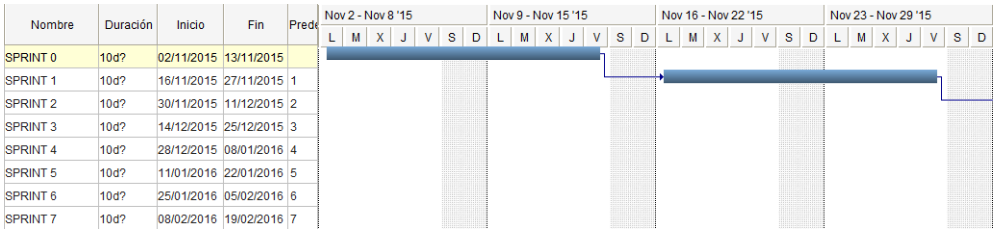
Plan de Publicación		
Sprint 0	**PBI_17**: Investigación Sobre la tecnología *FUSE*	5 PDH
	PBI_18: Prueba de concepto de un sistema *FUSE*	5 PDH
Sprint 1	**PBI_01**: Como usuario quiero compilar la aplicación utilizando la línea de comandos	3 PDH
	PBI_02: Como usuario, quiero montar los sistemas de ficheros en el punto de montaje a través de la línea de comandos para poder acceder a sus estructuras de archivos.	3 PDH
	PBI_04: Como usuario, quiero crear carpetas a través del punto de montaje	5 PDH
Sprint 2	**PBI_03**: Como usuario, quiero recibir las instrucciones sobre los parámetros que debo pasarle a la aplicación para que se ejecute correctamente	1 PDH

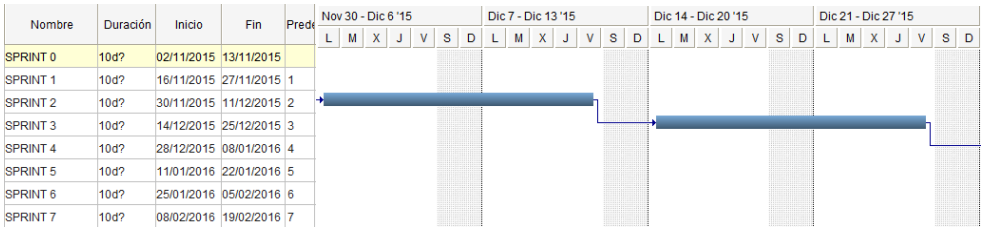
Plan de Publicación		
	PBI_05: Como usuario, quiero borrar carpetas de varios sistemas de ficheros a través del punto de montaje	5 PDH
	PBI_07: Como usuario, quiero navegar por las carpetas de los sistemas de ficheros montados a través del punto de montaje	5 PDH
Sprint 3	**PBI_06**: Como usuario, no quiero que se borren carpetas de varios sistemas de ficheros a través del punto de montaje, cuando estas tienen contenido	3 PDH
	PBI_08: Como usuario, quiero listar el contenido de una carpeta a través del punto de montaje	5 PDH
	PBI_09: Como usuario, quiero que se me indique, al realizar un listado del contenido de un directorio, si este no existe	3 PDH
Sprint 4	**PBI_10**: Como usuario, quiero copiar archivos a uno de los sistemas de ficheros montados a través del punto de montaje	8 PDH
	PBI_11: Como usuario, quiero recibir un mensaje de error si, al copiar un archivo al sistema de ficheros a través del punto de montaje, este no existe	3 PDH
Sprint 5	**PBI_12**: Como usuario quiero crear nuevos archivos en los sistemas de ficheros montados a través del punto de montaje	8 PDH
	PBI_13: Como usuario, quiero recibir un mensaje de error si, al crear un archivo, este ya existe	3 PDH
Sprint 6	**PBI_14**: Como usuario, quiero abrir archivos alojados en los sistemas de ficheros montados a través del punto de montaje	5 PDH

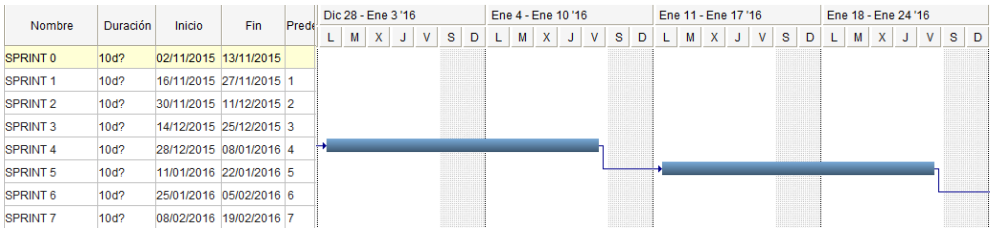
Plan de Publicación		
	PBI_15: Como usuario quiero borrar archivos alojados en los sistemas de ficheros montados a través del punto de montaje	5 PDH
Sprint 7	**PBI_16**: Como usuario quiero renombrar archivos y carpetas alojados en los sistemas de ficheros montados a través del punto de montaje.	8 PDH

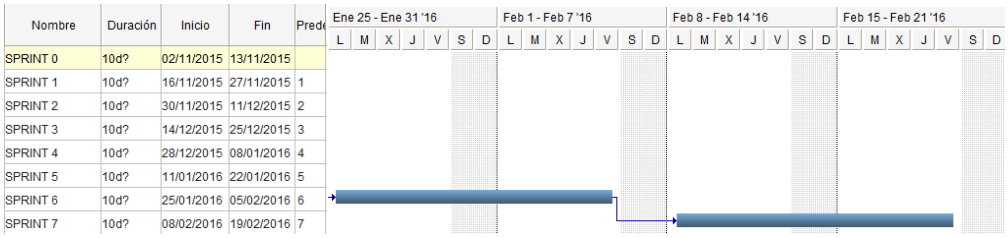
Tabla 91 Plan de Publicación

5.4 Diagrama de Gantt









6 Pruebas

En este capítulo, se mostrará un resumen de las sesiones de demo realizadas al final cada una de las iteraciones.

Es conveniente destacar la falta de referencias al Sprint 0. La razón se encuentra en que, en planificaciones ágiles, el primer sprint o Sprint 0, es considerado como un sprint de preparación. En él se preparan los entornos que van a ser necesarios para el desarrollo, se investiga la tecnología y se adelantan los posibles bloqueos que se van a encontrar durante el desarrollo. Salvo excepciones, las actividades realizadas en éste sprint no aportan valor demostrable de cara al *Product Manager* y los *Stake Holders* por lo que no se realiza una demo delante de ellos.

A continuación, se muestra el resultado de las siguientes demos

6.1 Sprint 0

6.1.1 Elementos del Backlog

Sprint 0	**PBI_17**: Investigación Sobre la tecnología FUSE	5 PDH
	PBI_18: Prueba de concepto de un sistema FUSE	5 PDH

Tabla 92 Sprint 0

6.2 Sprint 1

6.2.1 Elementos del Backlog

Sprint 1	**PBI_01**: Como usuario quiero compilar la aplicación utilizando la línea de comandos	3 PDH

	PBI_02: Como usuario, quiero montar los sistemas de ficheros en el punto de montaje a través de la línea de comandos para poder acceder a sus estructuras de archivos.	3 PDH
	PBI_04: Como usuario, quiero crear carpetas a través del punto de montaje	5 PDH

Tabla 93 Sprint 1

6.2.2 Pruebas realizadas

PBI_01

En este PBI, se prueba el script Make creado para la compilación del sistema. Es necesario disponer del código fuente. Para el desarrollo de esta prueba se utilizará como espacio de trabajo la carpeta workspace/fuse-project.

- **Escenario 1:** Con el código fuente disponible, se navega a la carpeta src y se ejecuta el comando make.

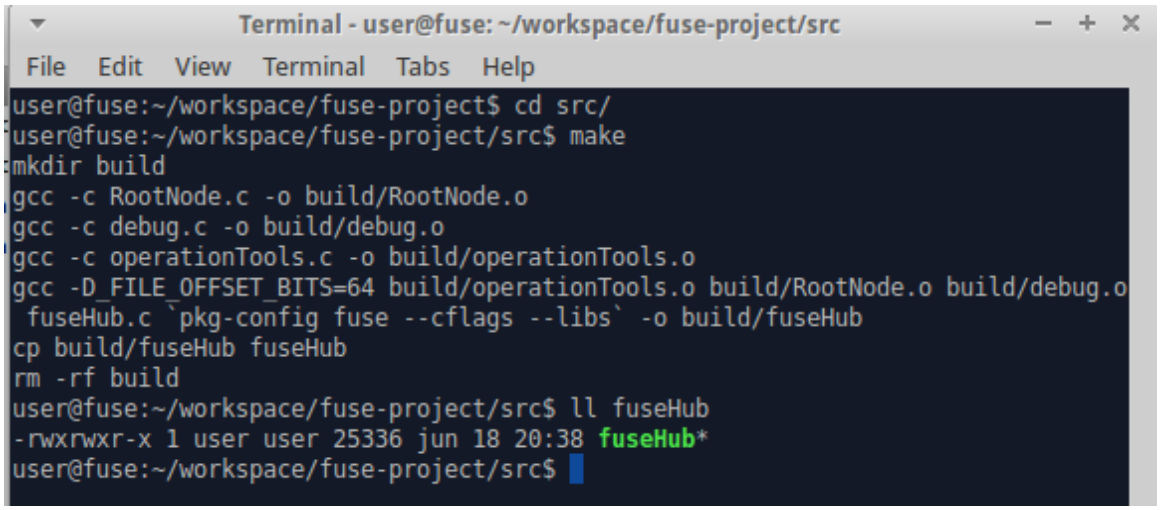


Ilustración 12 Resultado PBI_01

Como resultado, se obtendrá un ejecutable llamado fuseHub.

PBI_02

Para la ejecución de los siguientes escenarios es necesario disponer el ejecutable del sistema y un fichero de texto en el que se encuentren las rutas de los sistemas de ficheros, ya montados, que se desean montar en el punto de montaje.

- **Escenario 1**: El sistema recibe, como primer parámetro la ruta del punto de montaje y, como segundo parámetro, la ruta del fichero que contiene las rutas de los sistemas de ficheros.

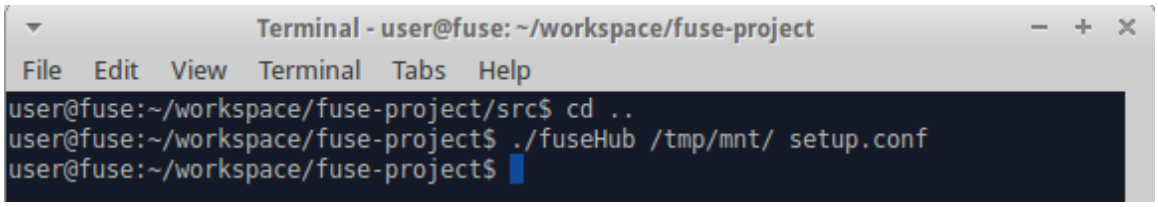


Ilustración 13 Resultado PBI_02

Al ejecutar el comando, los sistemas de ficheros quedarán montados en el punto de montaje.

PBI_04

Para el desarrollo de esta prueba, ejecutaremos la aplicación con cuatro sistemas de ficheros vacíos montados en el punto de montaje.

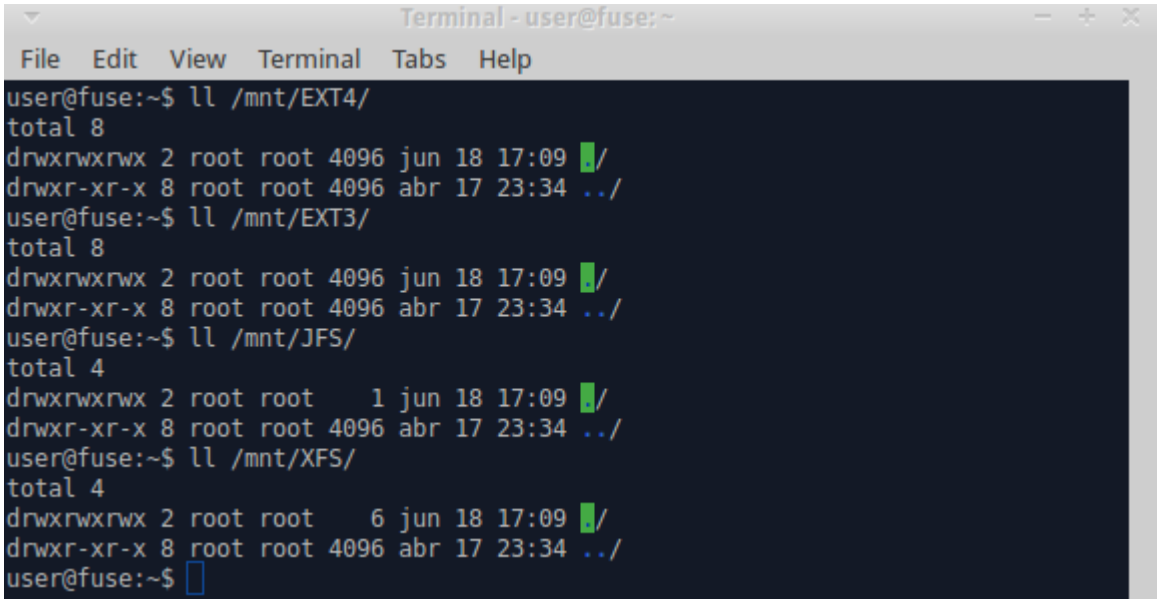


Ilustración 14 Precondiciones PBI_04

Al ejecutar una orden de listado, el resultado es el siguiente:

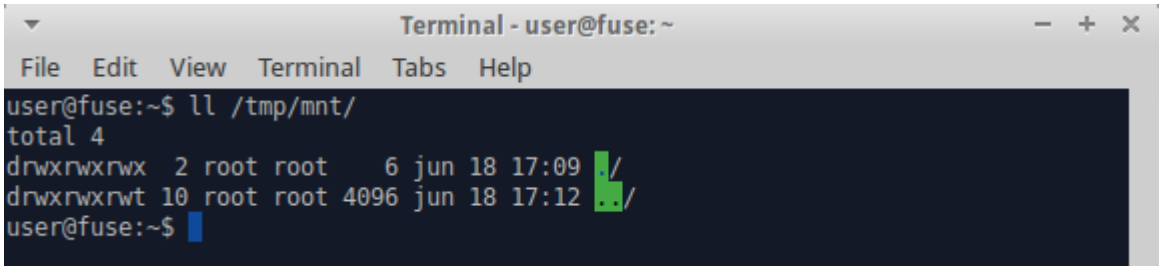


Ilustración 15 Ejecución PBI_04

Al ejecutar un comando de creación de carpetas con una ruta relativa al punto de montaje, se creará una carpeta en cada uno de los sistemas de ficheros montados.

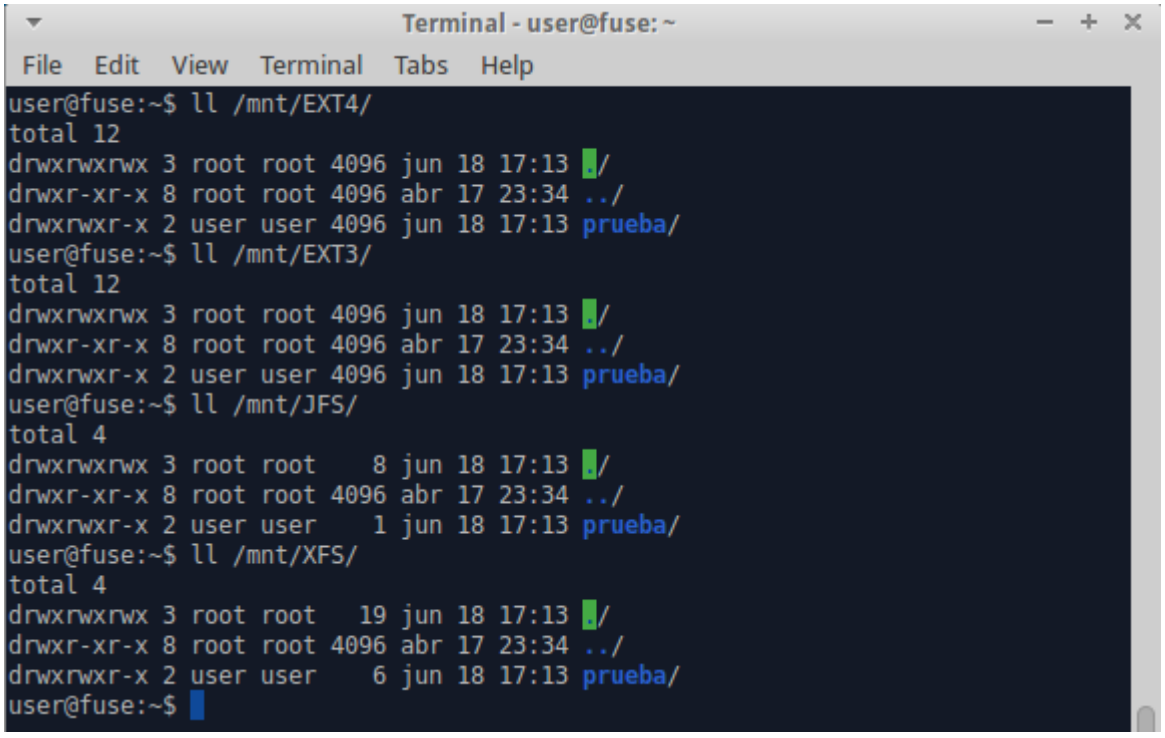


Ilustración 16 Primer Resultado PBI_04

Listando el contenido del punto de montaje, sólo aparece una carpeta.

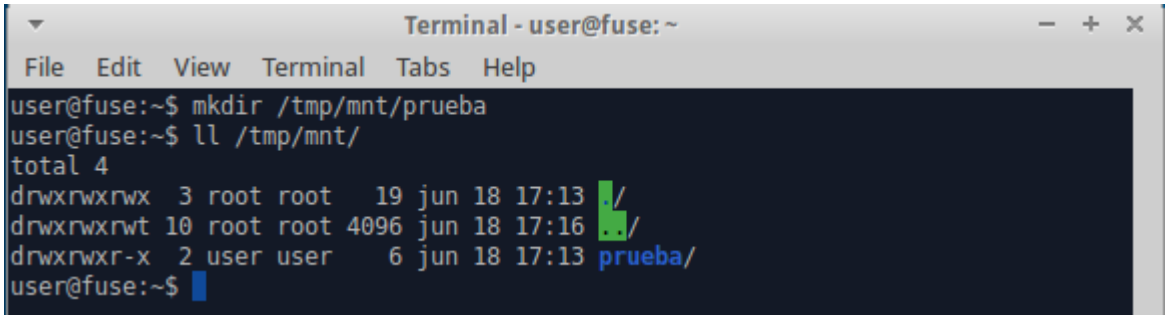


Ilustración 17 Segundo Resultado PBI_04

6.3 Sprint 2

6.3.1 Elementos del Backlog

Sprint 2	**PBI_03**: Como usuario, quiero recibir las instrucciones sobre los parámetros que debo pasarle a la aplicación para que se ejecute correctamente	1 PDH
	PBI_05: Como usuario, quiero borrar carpetas de varios sistemas de ficheros a través del punto de montaje	5 PDH
	PBI_07: Como usuario, quiero navegar por las carpetas de los sitemas de ficheros montados a través del punto de montaje	5 PDH

Tabla 94 Sprint 2

6.3.2 Pruebas realizadas

PBI_03

Para realizar las pruebas relacionadas con este PBI, es necesario tener compilado el código del sistema.

- **Escenario 1**: En este caso, la aplicación se ejecuta con un número adecuado de parámetros y con un número de sistemas de ficheros permitidos.

  Al ejecutar la aplicación, se obtiene la siguiente salida:

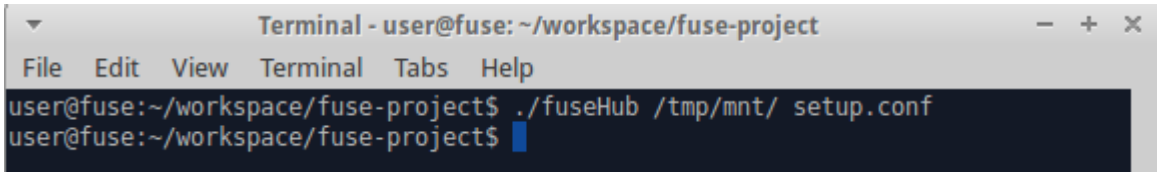


Ilustración 18 Primer Escenario PBI_03

- **Escenario 2**: La aplicación es ejecutada con un número incorrecto de parámetros. Se dan dos casos.

    - La aplicación se ejecuta con menos parámetros de los necesarios.

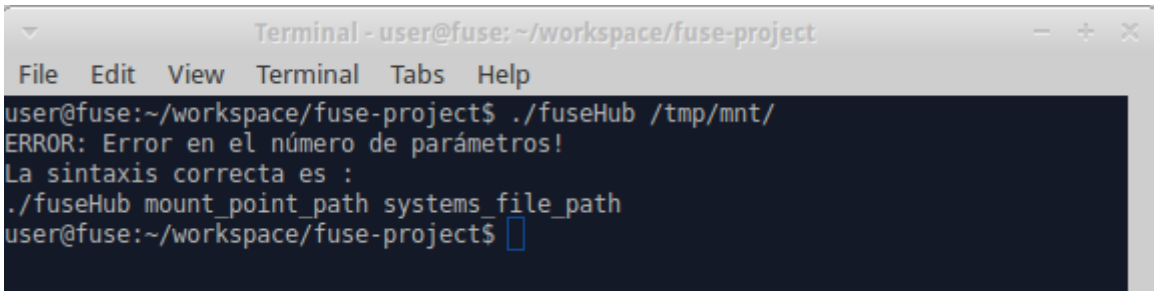


Ilustración 19 Precondiciones Segundo Escenario PBI_03

    - La aplicación se ejecuta con más parámetros de los necesarios.

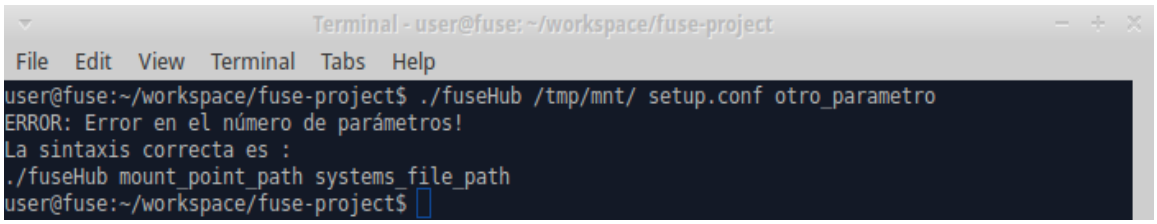


Ilustración 20 Resultado Segundo Escenario PBI_03

- **Escenario 3**: En el fichero con las rutas de los sistemas de ficheros montados, hay menos que el mínimo permitido.

    Ejecutando la aplicación, se obtiene la siguiente salida:

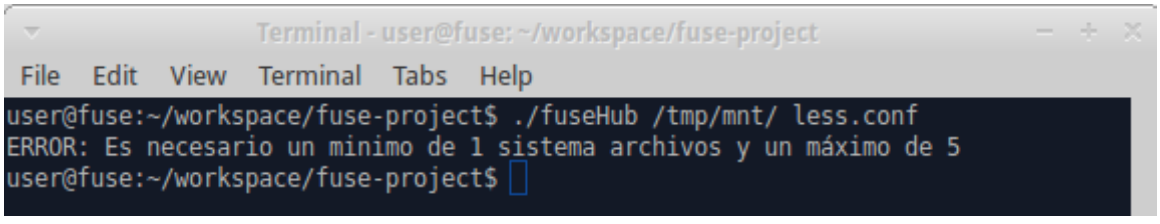


Ilustración 21 Resultado Tercer Escenario PBI_03

- **Escenario 4**: En el fichero de rutas a sistemas de ficheros, hay más entradas de las permitidas.

Ejecutando la aplicación, se obtiene la siguiente salida:

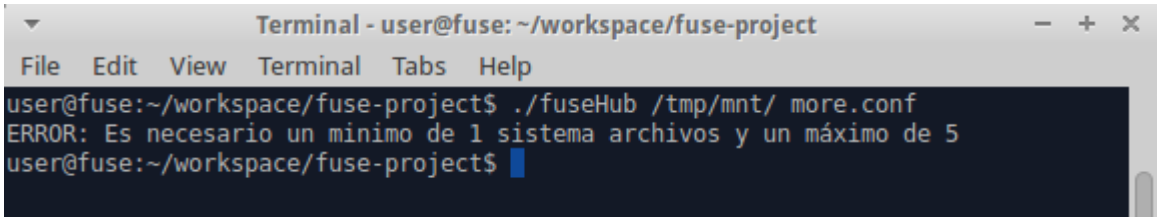


Ilustración 22 Resultado Cuarto Escenario PBI_03

PBI_05

Partimos de una situación en la que tenemos cuatro sistemas de ficheros montados en el punto de montaje.

- **Escenario 1**: En este caso, la carpeta que se desea eliminar se encuentra, al menos, en uno de los sistemas de ficheros montados.

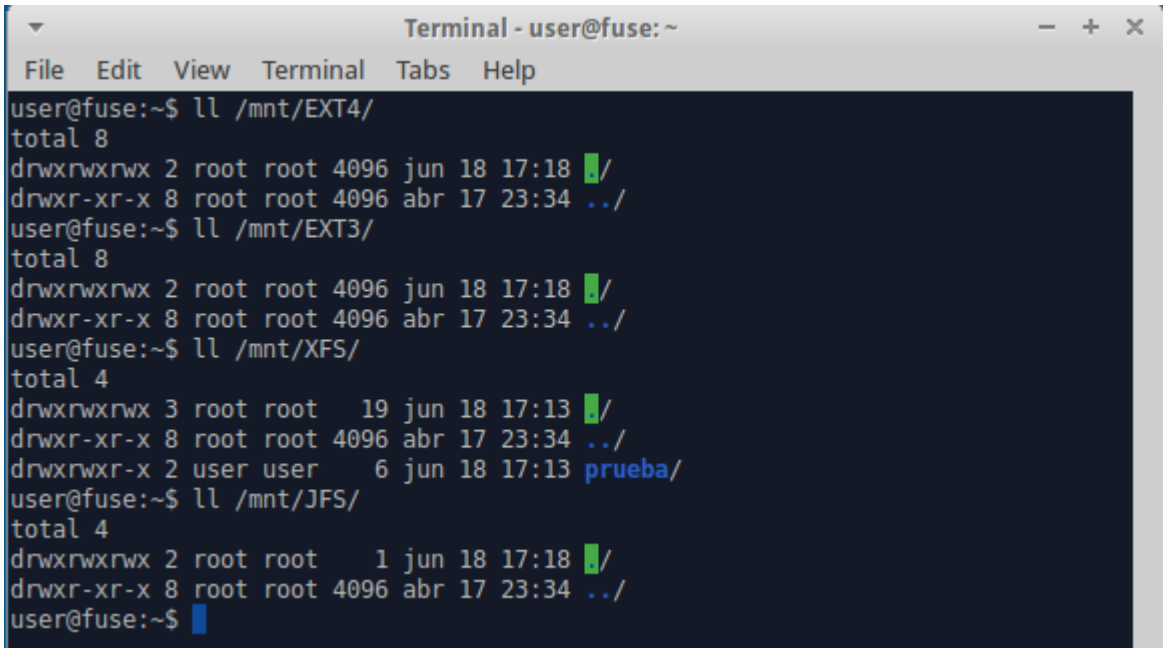


Ilustración 23 Precondiciones Primer Escenario PBI_05

Al ejecutar el comando de borrado de carpetas, pasándole como parámetro la ruta de la carpeta relativa al punto de montaje, esta se borra del sistema de ficheros en el que se encuentra y no se muestra ningún error.

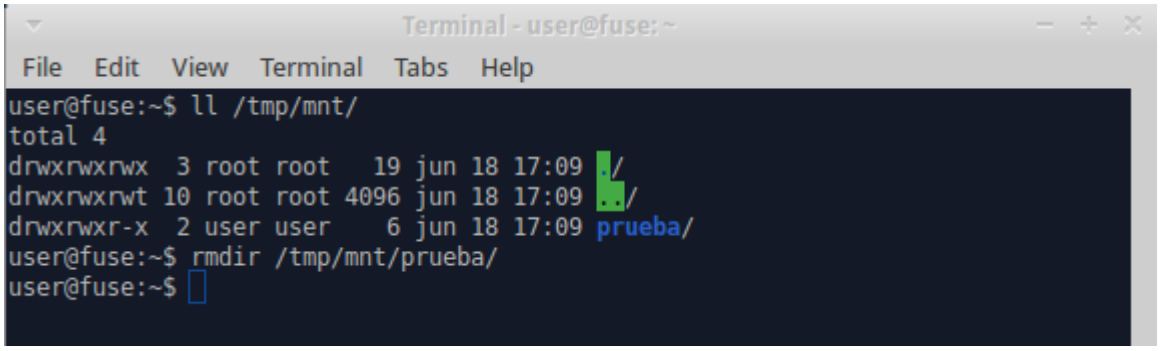


Ilustración 24 Primer Resultado Del Primer Escenario PBI_05

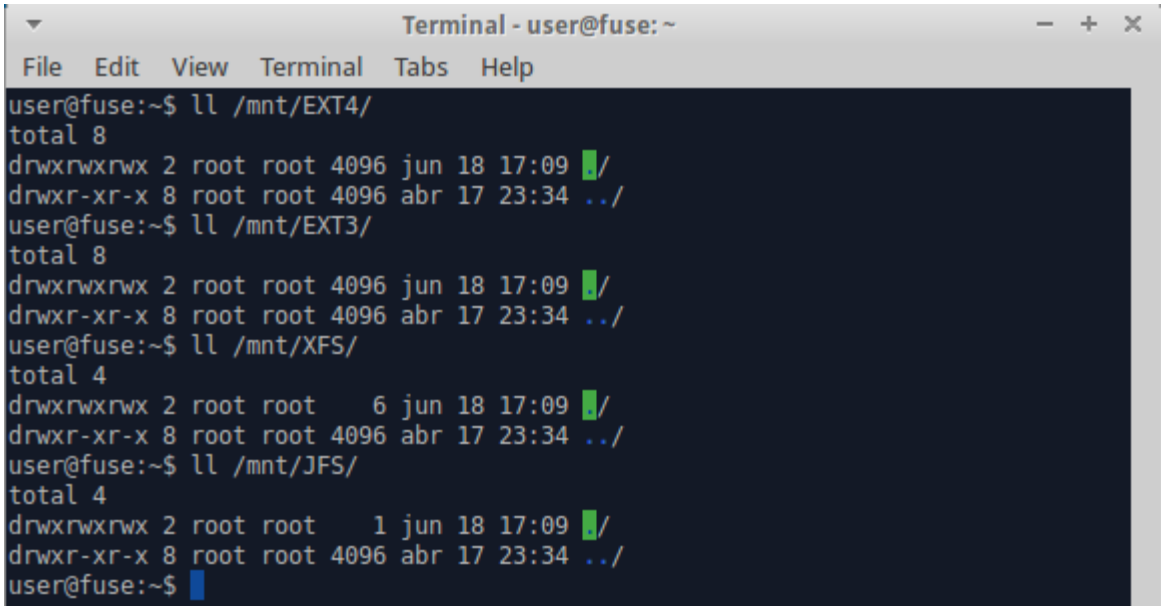


Ilustración 25 Segundo Resultado Del Primer Escenario PBI_05

- **Escenario 2**: La carpeta, en este caso, se encuentra en todos los sistemas de ficheros montados en el punto de montaje.

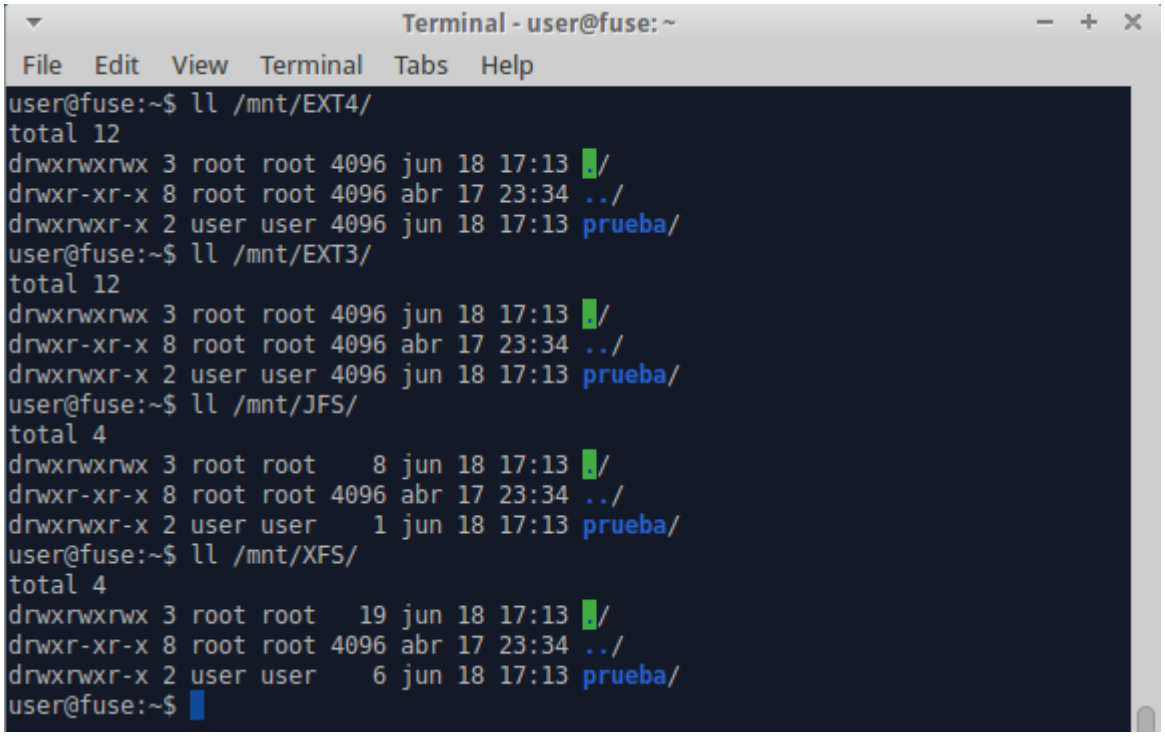


Ilustración 26 Precondiciones Segundo Escenario PBI_05

Ejecutando el comando de borrado de carpetas, pasándole como parámetro la ruta de la carpeta relativa al punto de montaje, esta se borra de todos los sistema de ficheros en el que se encuentra y no se muestra ningún error.

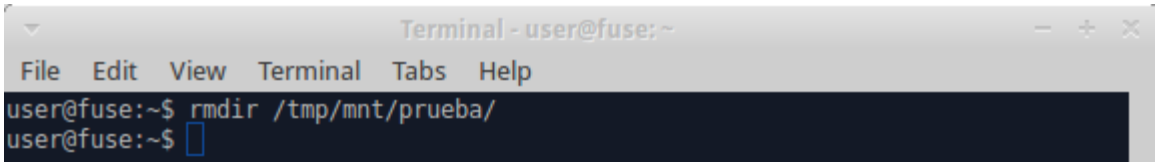


Ilustración 27 Primer Resultado Del Segundo Escenario PBI_05

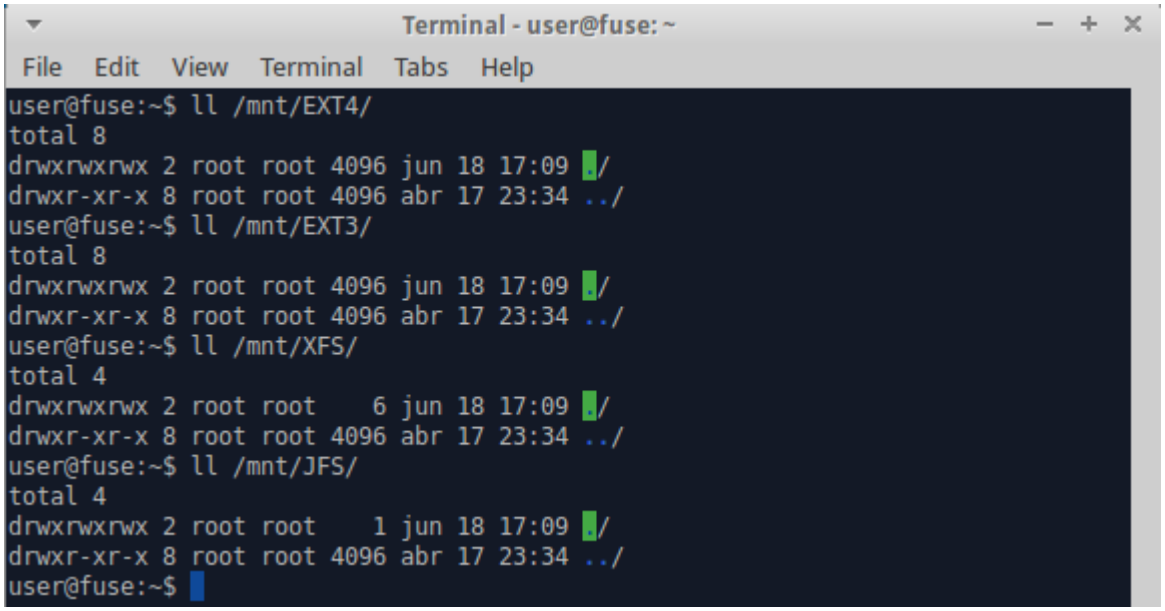


Ilustración 28 Segundo Resultado Del Primer Escenario PBI_05

- **Escenario 3**: El último escenario a probar para este PBI abarca la posibilidad de que se intente borrar una carpeta que no se encuentra en ninguno de los sistemas de ficheros montados. Dado el siguiente contenido dentro de los sistemas de ficheros:

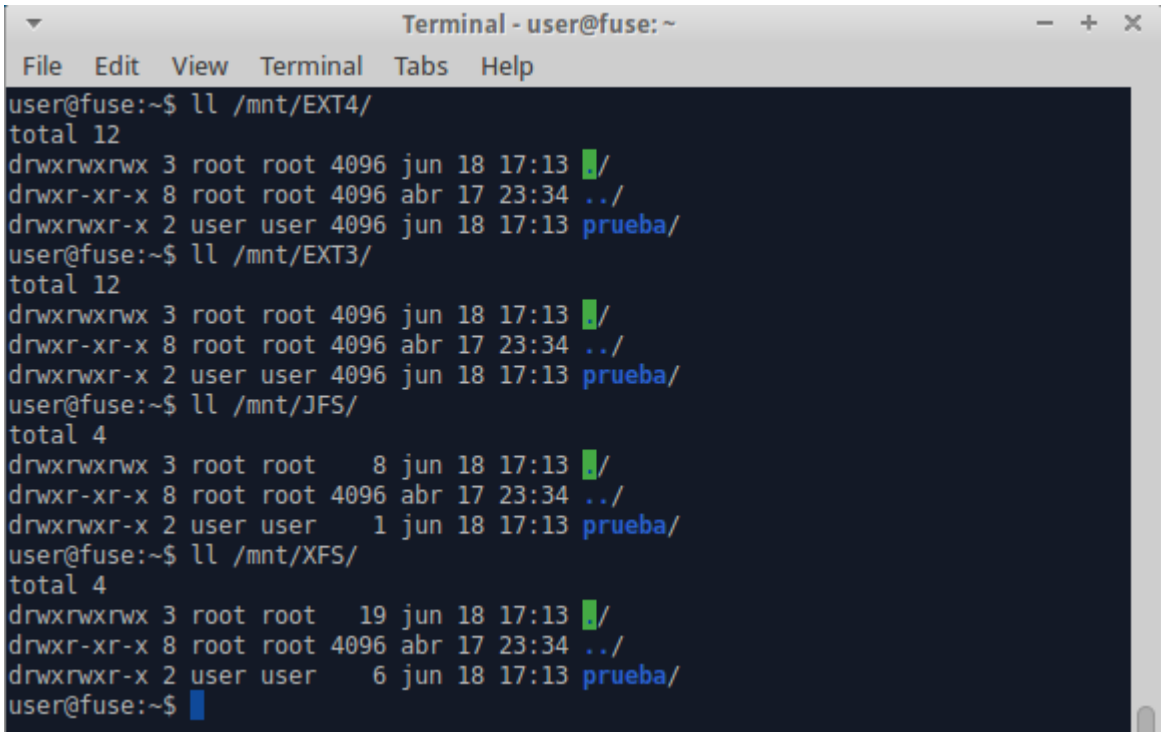


Ilustración 29 Precondiciones Tercer Escenario PBI_05

Al ejecutar el comando de borrado con una ruta relativa al punto de montaje que no existe, se muestra el siguiente error:

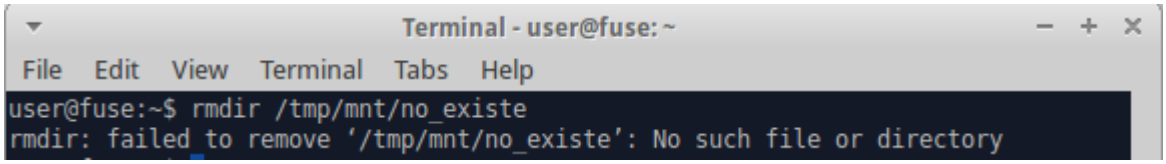


Ilustración 30 Resultado Tercer Escenario PBI_05

PBI_07

Para la ejecución de los siguientes escenarios de pruebas, se partirá de que el siguiente contenido de los sistemas de ficheros es accesible a través del punto de montaje:

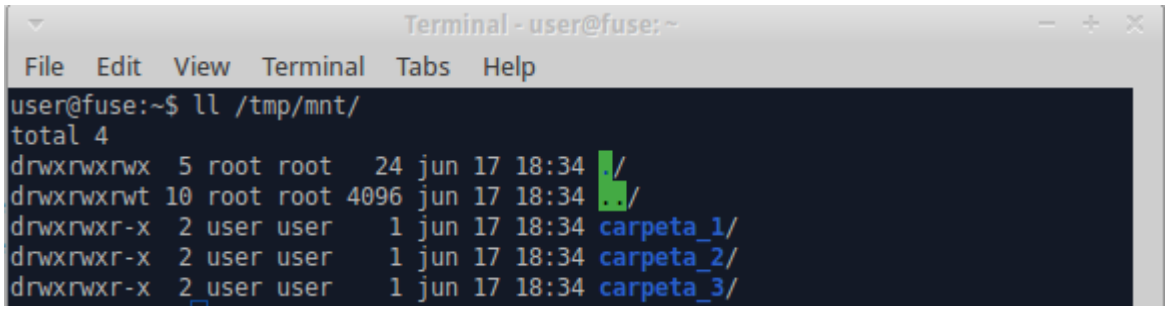


Ilustración 31 Precondiciones PBI_07

- **Escenario 1**: En este escenario, se reproduce la situación en la que el usuario desea acceder a una de las carpetas alojadas en, al menos uno de los sistemas de ficheros montados en el punto de montaje. Al ejecutar el comando de acceso a una carpeta existente, la ruta en la que se encuentra el usuario sigue siendo relativa al punto de montaje.

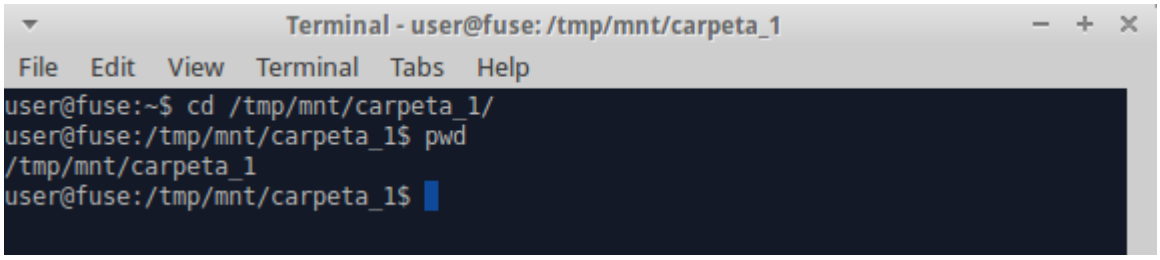


Ilustración 32 Resultado Primer Escenario PBI_07

- **Escenario 2**: En este escenario, se reproduce el comportamiento de la aplicación al intentar acceder a una ruta no existente en ninguno de los sistemas de ficheros montados. De esta forma, al intentar acceder a la ruta de una carpeta que no se encuentra dentro del contenido mostrado, se produce el siguiente error:

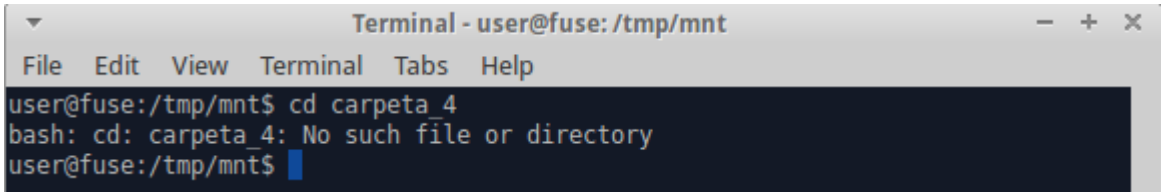


Ilustración 33 Resultado Segundo Escenario PBI_07

6.4 Sprint 3

6.4.1 Elementos del Backlog

Sprint 3	**PBI_06**: Como usuario, no quiero que se borren carpetas de varios sistemas de ficheros a través del punto de montaje, cuando estas tienen contenido	3 PDH
	PBI_08: Como usuario, quiero listar el contenido de una carpeta a través del punto de montaje	5 PDH
	PBI_09: Como usuario, quiero que se me indique, al realizar un listado del contenido de un directorio, si este no existe	3 PDH

Tabla 95 Sprint 3

6.4.2 Pruebas realizadas

PBI_06

Para realizar las pruebas, montaremos 3 sistemas de ficheros en el punto de montaje. En uno de ellos, el contenido de uno de los puntos de montaje es el siguiente:

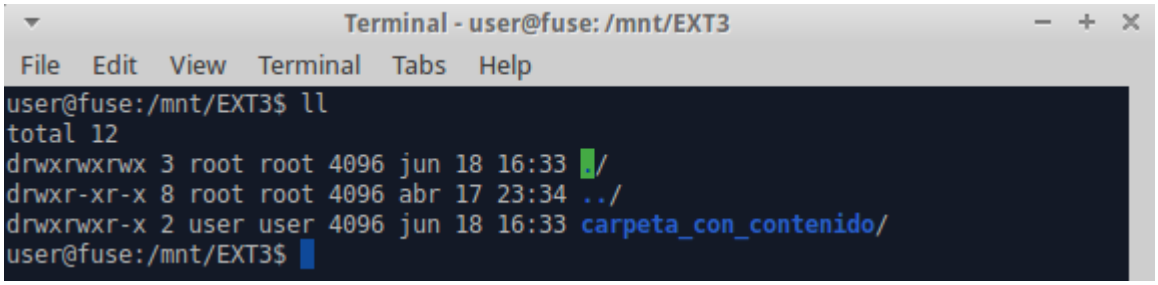


Ilustración 34 Primera Precondición PBI_06

Y dentro de la carpeta "*carpeta_con_contenido*", se encuentra el fichero "*fichero.txt*".

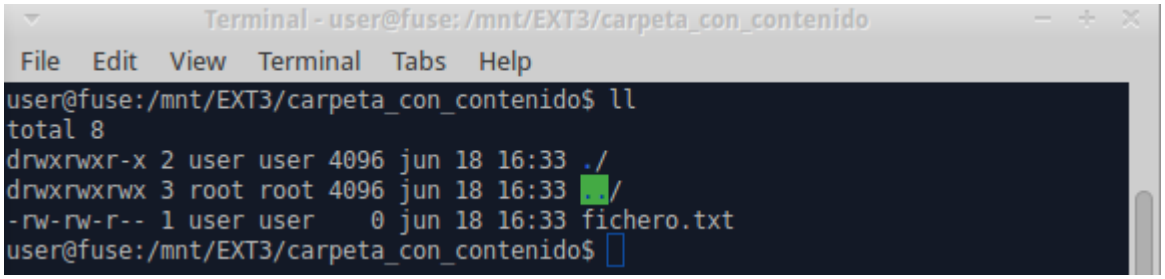


Ilustración 35 Segunda Precondición PBI_06

- **Escenario 1**: En este escenario, probaremos el caso en el que la carpeta con contenido sólo existe en uno de los sistemas de ficheros montados.

  Ejecutando el comando de borrado de carpetas sobre la ruta de la carpeta, relativa al punto de montaje, obtenemos el siguiente mensaje de error:

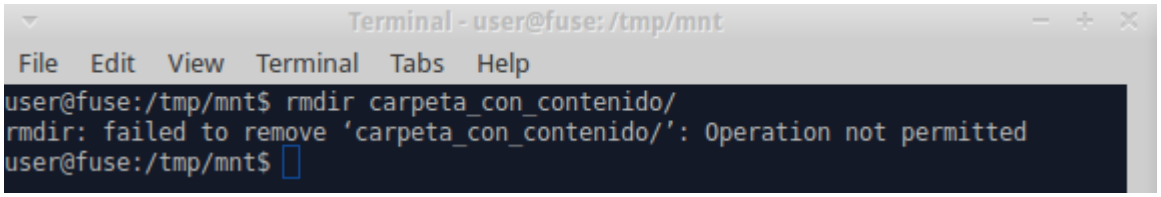


Ilustración 36 Resultado Primer Escenario PBI_06

  Tras la ejecución del comando ni la carpeta ni su contenido han sido borrados.

- **Escenario 2**: Para este escenario, la carpeta con contenido existe en todos los sistemas de ficheros montados. Dicha carpeta, sólo posee contenido en uno de los sistemas de ficheros.

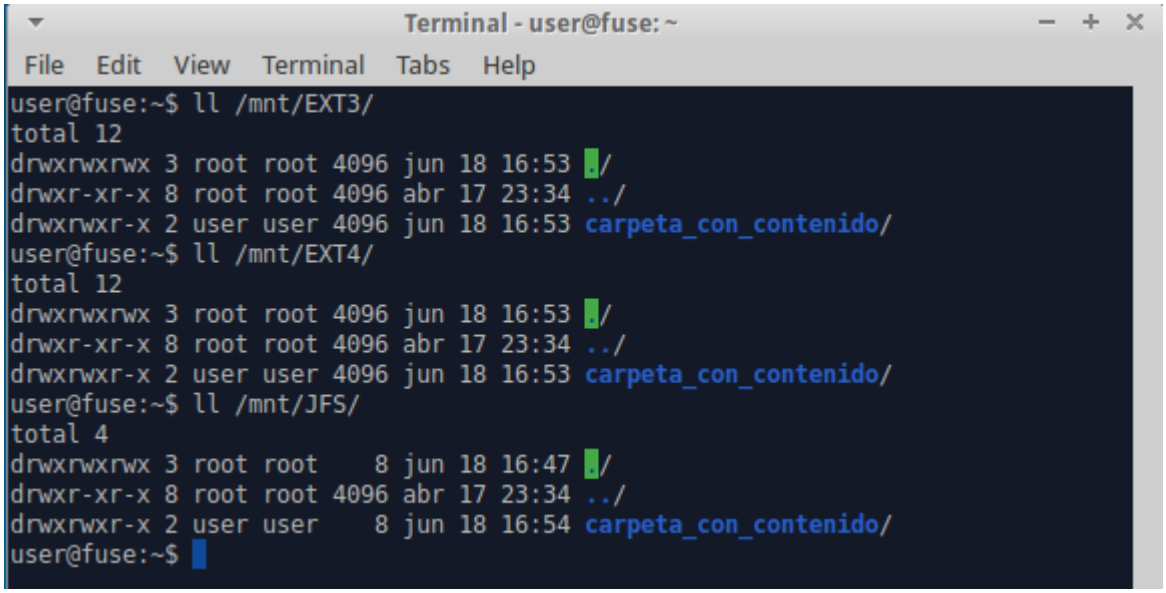


Ilustración 37 Resultado Segundo Escenario PBI_06

Ejecutando el comando de borrado de carpetas sobre la ruta de la carpeta, relativa al punto de montaje, obtenemos el siguiente mensaje de error:

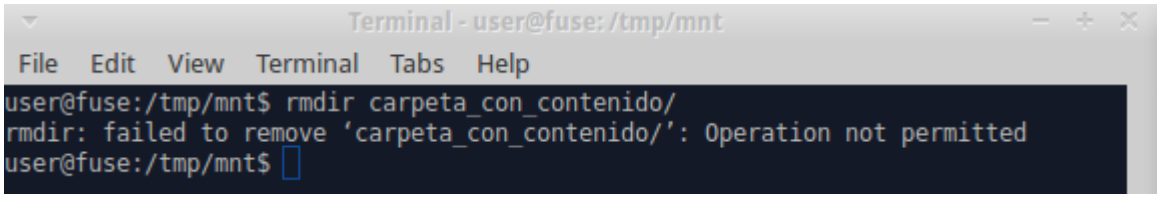


Ilustración 38 Segundo Resultado Del Segundo Escenario PBI_06

Tras la ejecución del comando, ninguna de las carpetas ha sido borrada. De la misma forma, el contenido de la carpeta tampoco ha sido afectado.

- **Escenario 3**: En este caso, la ruta de la carpeta existe en todos los sistemas de ficheros y, en todos ellos posee contenido.

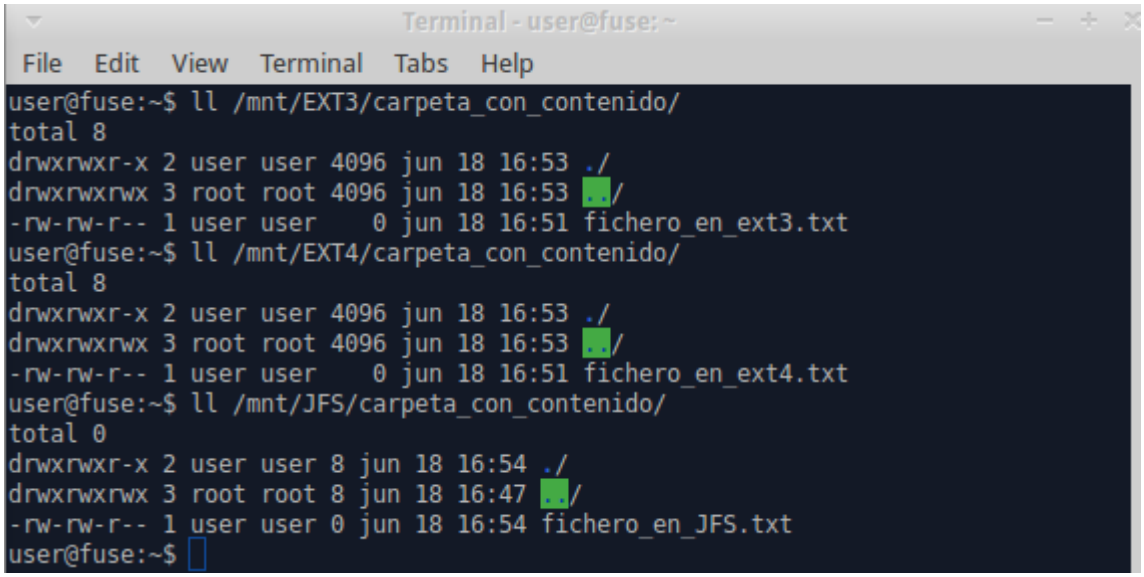


Ilustración 39 Primer Resultado Del Tercer Escenario PBI_06

Ejecutando el comando de borrado de carpetas sobre la ruta de la carpeta, relativa al punto de montaje, obtenemos el siguiente mensaje de error:

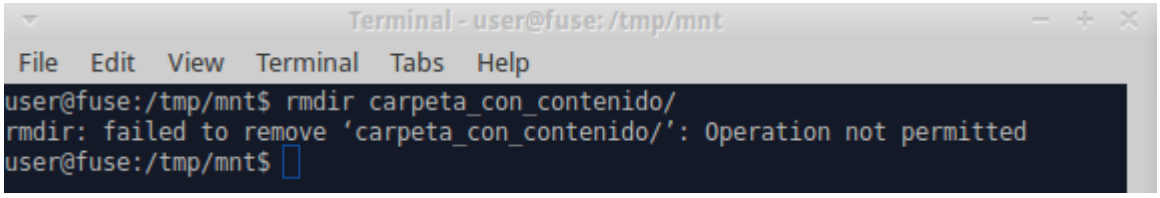


Ilustración 40 Segundo Resultado Del Tercer Escenario PBI_06

Una ejecutado el comando, ni las carpetas ni su contenido han sido borrados.

PBI_08

En este caso, se va a probar el listado de contenidos dentro del punto de montaje. El punto clave a tener en cuenta para realizar la prueba es la multiplicidad de las rutas relativas al punto de montaje en los diferentes sistemas de ficheros.

- **Escenario 1**: Para este escenario, creamos una carpeta en uno de los sistemas de ficheros montados.

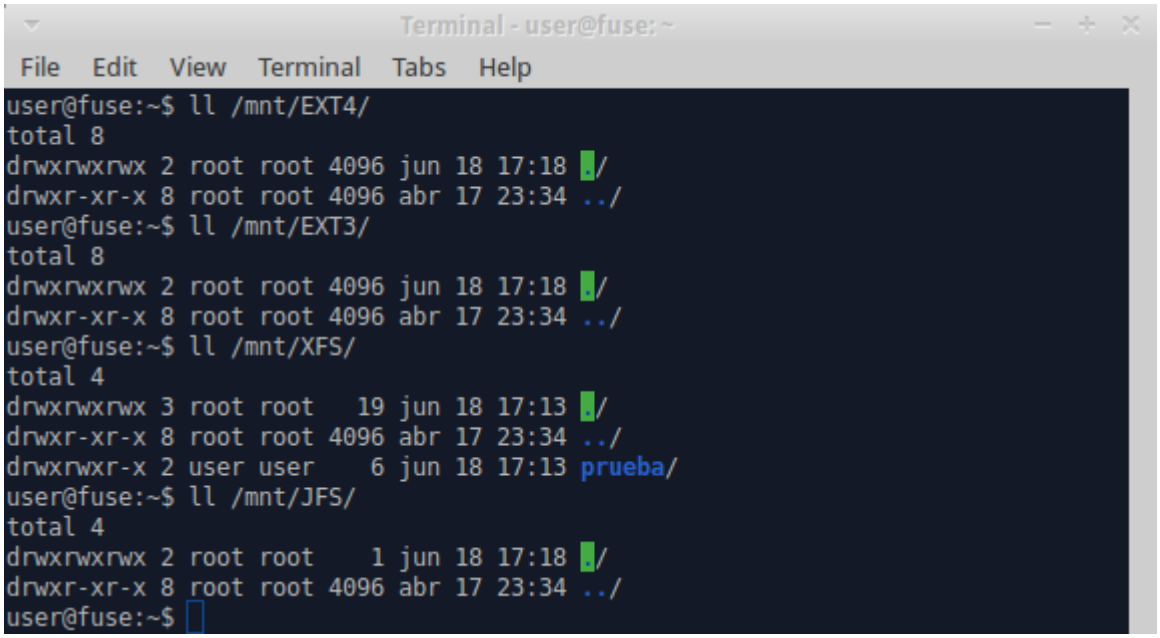


Ilustración 41 Precondiciones Primer Escenario PBI_08

Ejecutamos un comando para listar el contenido del punto de montaje.

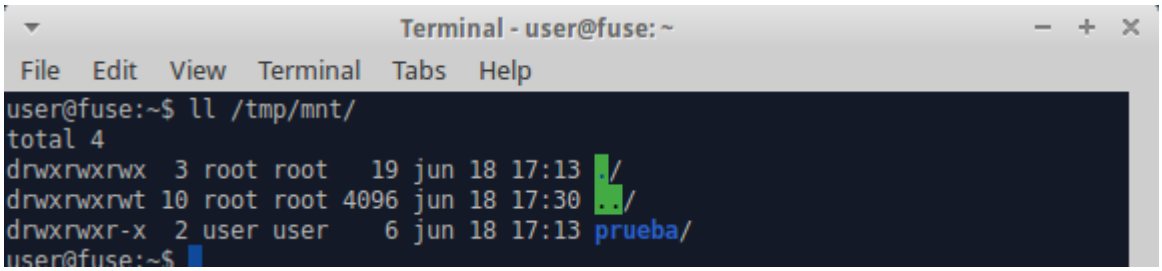


Ilustración 42 Resultado Primer Escenario PBI_08

Una vez ejecutado, el comando muestra la única carpeta existente.

- **Escenario 2**: Para probar este escenario, una carpeta con el mismo nombre en cada uno de los sistemas de ficheros montados. Es necesario que la ruta de cada una de las carpetas creadas, con respecto a su sistema de ficheros, sea igual.

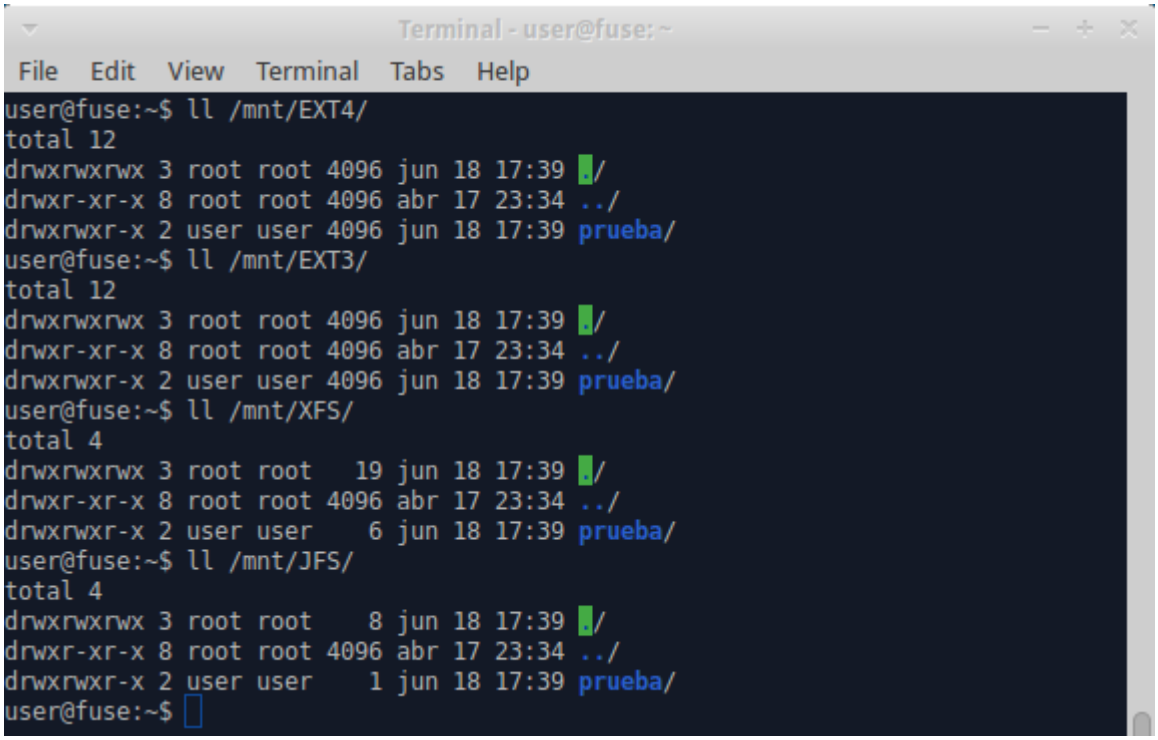


Ilustración 43 Precondiciones Segundo Escenario PBI_08

Ejecutando el comando de listado de contenidos, se muestra como una sola carpeta, sin duplicados.

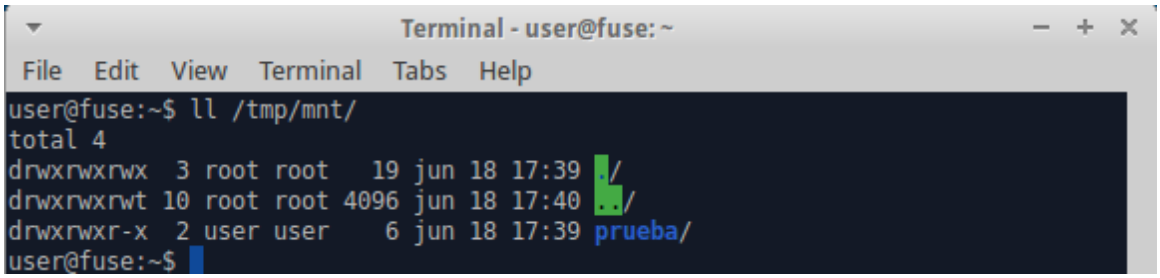


Ilustración 44 Resultados Segundo Escenario PBI_08

PBI_09

- **Escenario 1**: En este caso, se reproducirá el caso en el que el usuario intenta listar el contenido de una ruta relativa al punto de montaje que no existe. El contenido de los sistemas de ficheros será el siguiente:

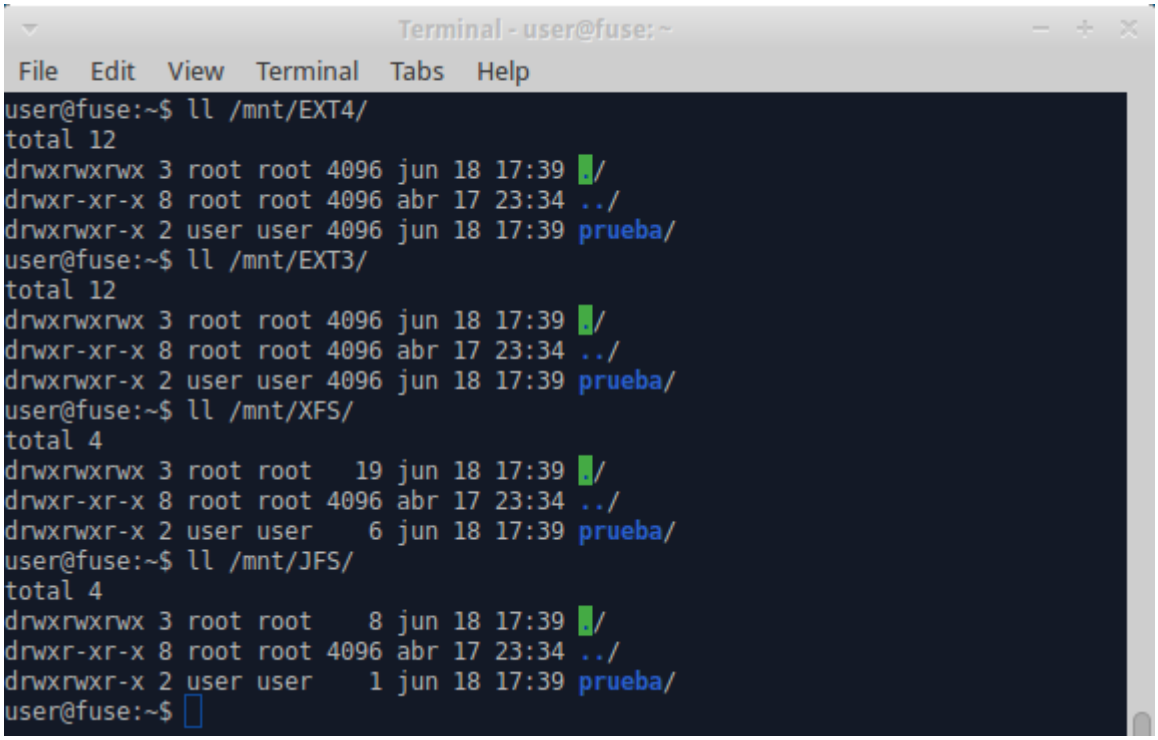


Ilustración 45 Precondiciones PBI_09

Ejecutando el comando de listado sobre una ruta que no existe, se obtiene el siguiente error:

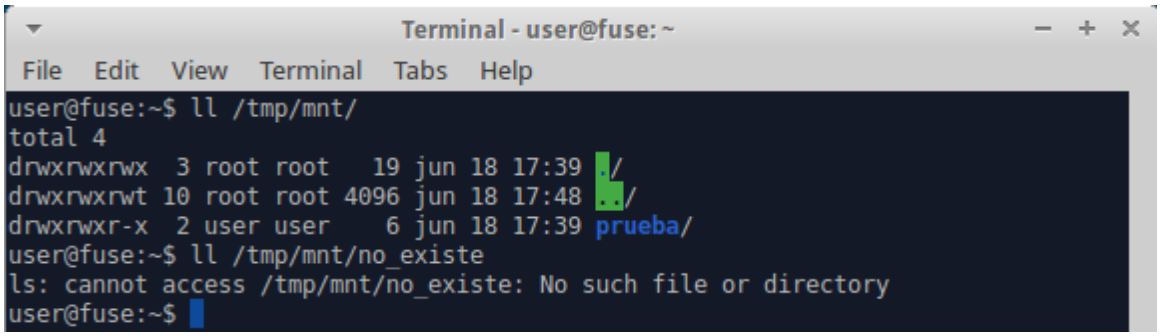


Ilustración 46 Resultado PBI_09

6.5 Sprint 4

6.5.1 Elementos del Backlog

Sprint 4	**PBI_10**: Como usuario, quiero copiar archivos a uno de los sistemas de ficheros montados a través del punto de montaje	8 PDH

	PBI_11: Como usuario, quiero recibir un mensaje de error si, al copiar un archivo al sistema de archivos a través del punto de montaje, este no existe	3 PDH

Tabla 96 Sprint 4

6.5.2 Pruebas realizadas

PBI_10

Para la prueba de este *PBI*, se va a trabajar con tres sistemas de ficheros diferentes. El contenido de los mismos será:

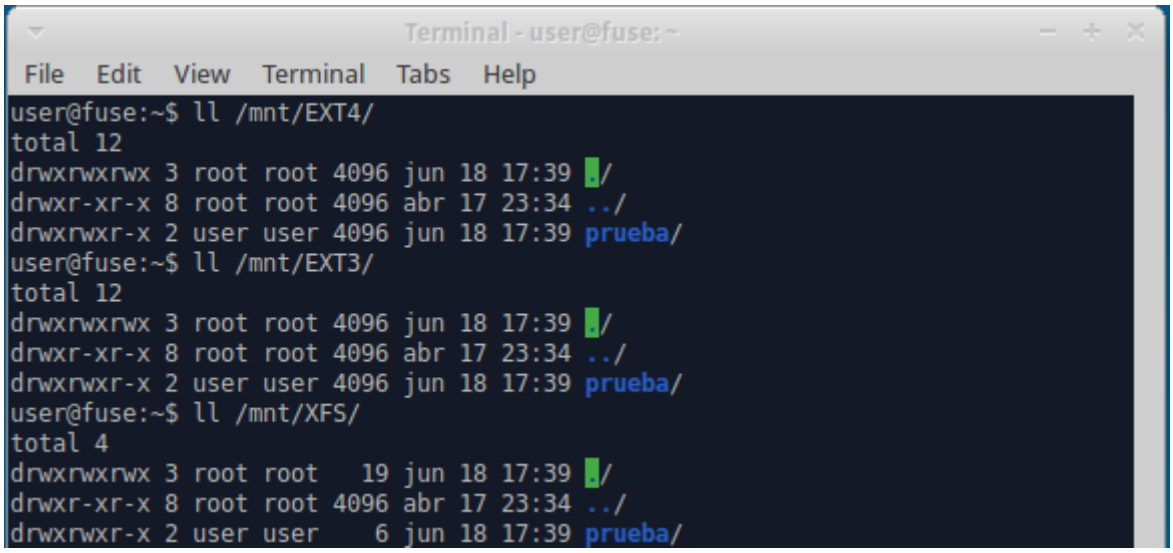


Ilustración 47 Precondición PBI_10

- **Escenario 1**: Este escenario reproduce la copia de un fichero a los sistemas de ficheros montados a través del punto de montaje.

  Cuando ejecutamos el comando pasando, como parámetro la ruta de un fichero existente, este es copiado a uno de los sistemas de ficheros y es visible ejecutando el comando de listado con la ruta en la que se ha copiado.

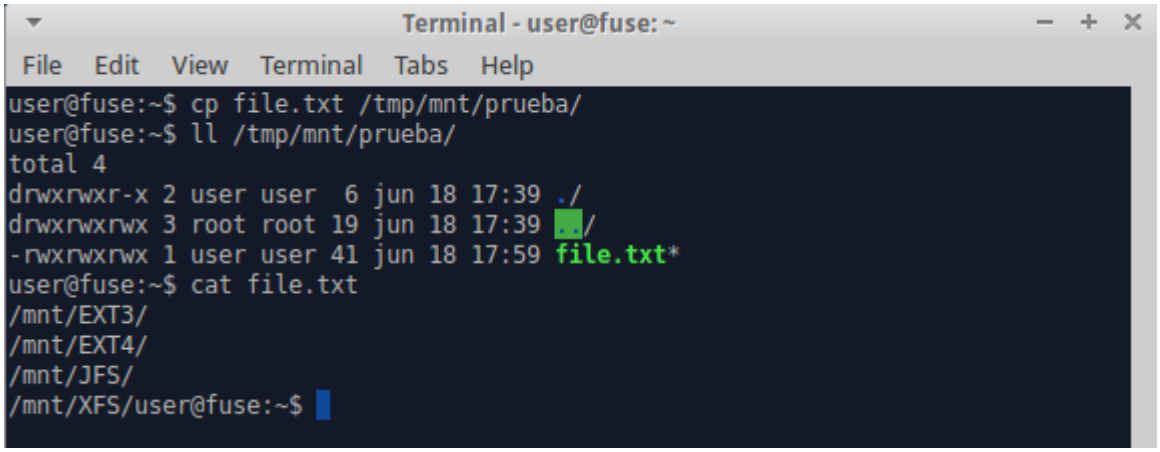


Ilustración 48 Resultado Primer Escenario PBI_10

- **Escenario 2**: Probaremos que el sistema elije los sistemas de ficheros en los que se almacenan los ficheros utilizando Round Robin. Para ello, vamos a copiar 4 ficheros.

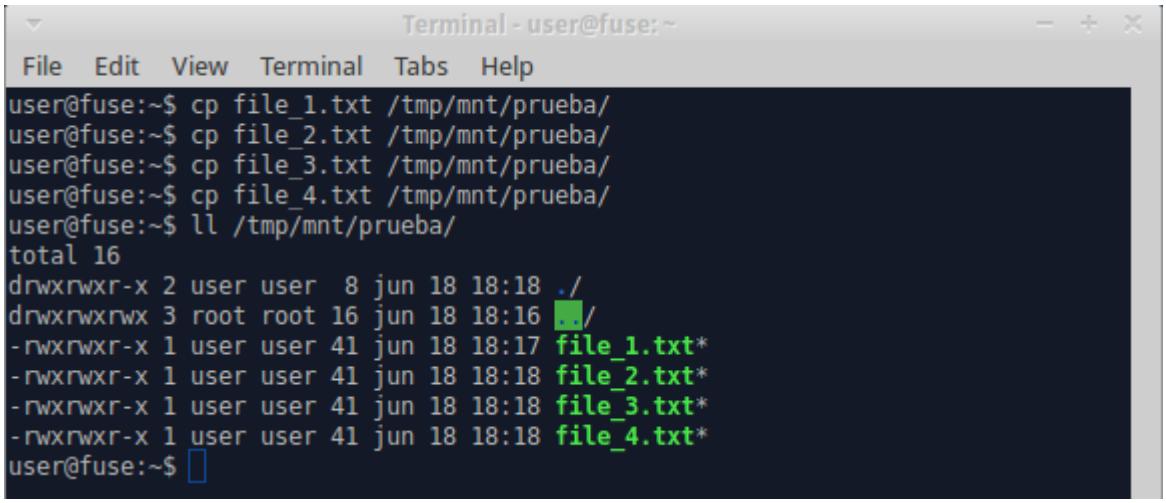


Ilustración 49 Resultado Segundo Escenario PBI_10

Como resultado, los tres primeros son copiados a diferentes sistemas de ficheros y el cuarto, al mismo que al que se copió el primero.

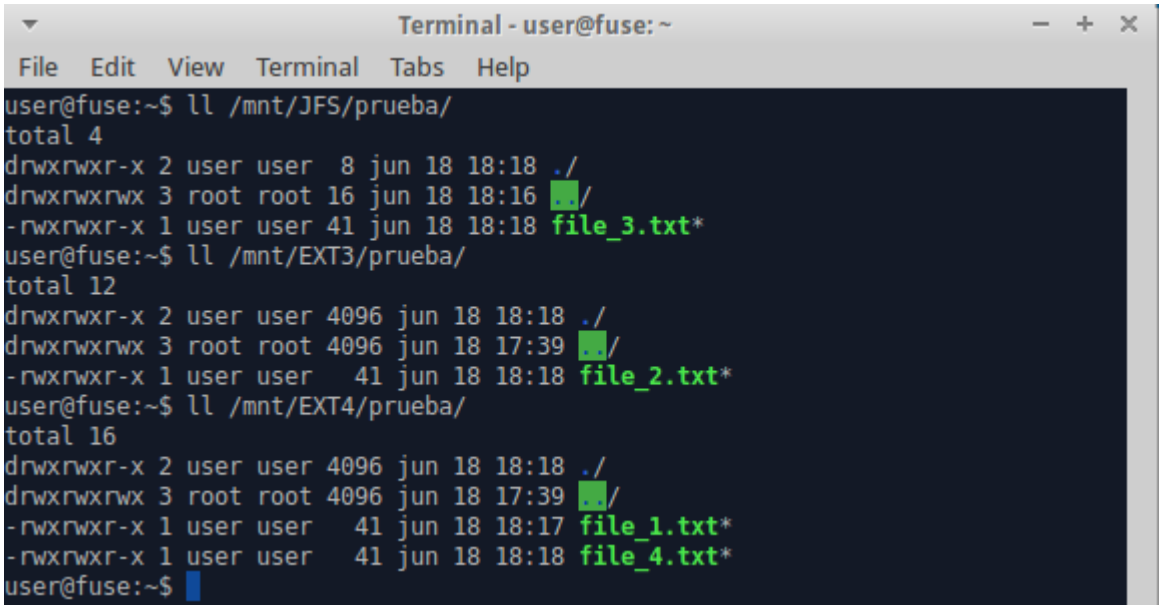


Ilustración 50 Segundo Resultado Del Segundo Escenario PBI_10

PBI_11

Este PBI cubre la gestión de los errores en el momento del copiado. Para ello, se va a manejar el siguiente esquema de sistemas de ficheros y contenidos:

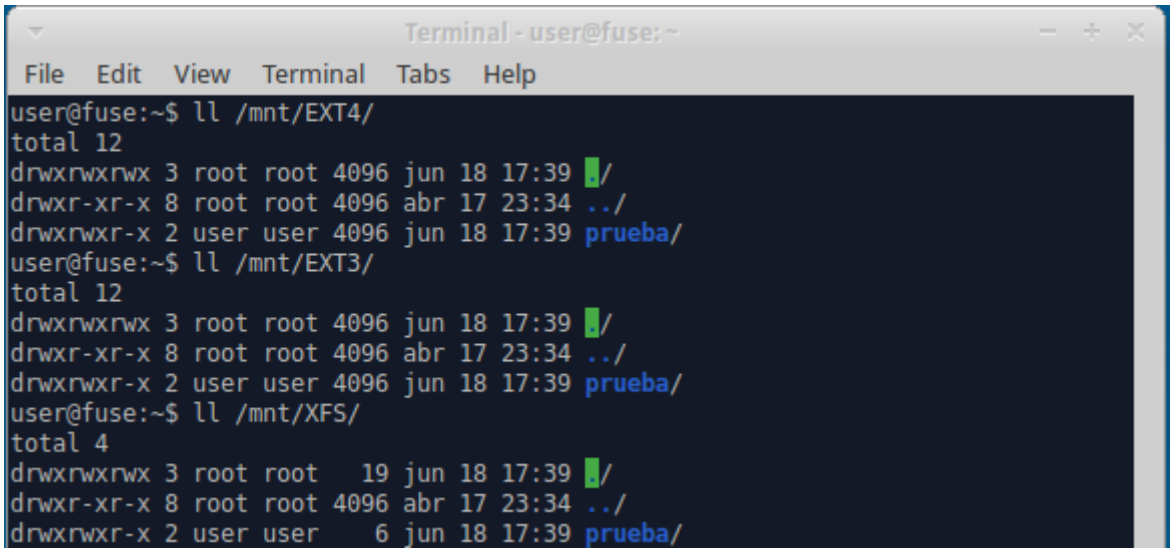


Ilustración 51 Precondición PBI_11

- **Escenario 1**: En este escenario, la ruta de destino del fichero no es visible desde el punto de montaje. Al ejecutar el comando de copiado con una ruta de destino inexistente, obtenemos el siguiente error:

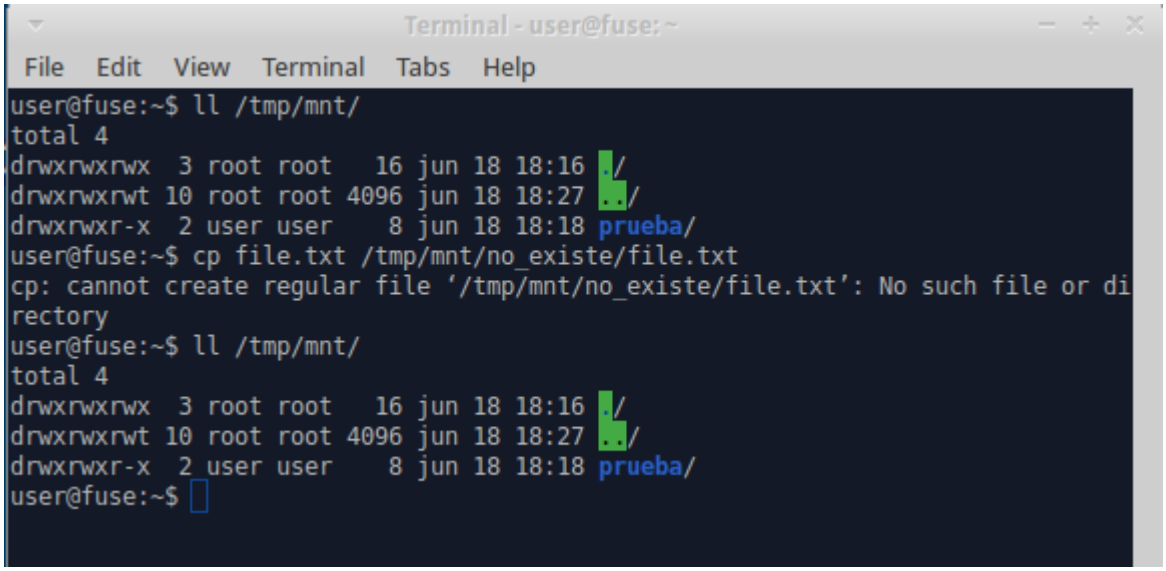


Ilustración 52 Resultado Primer Escenario PBI_11

- **Escenario 2**: En este caso, es la ruta de origen la que no existe. Al ejecutar el comando, se obtiene el siguiente error.

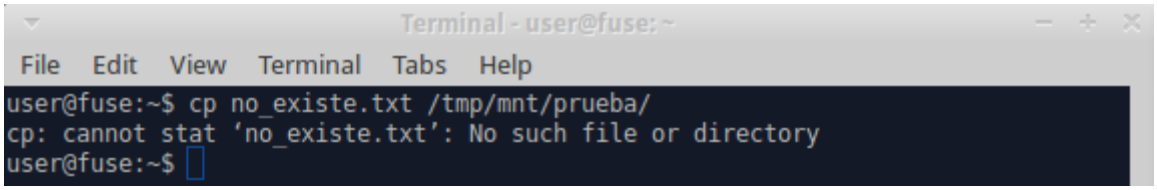


Ilustración 53 Resultado Segundo Escenario PBI_11

6.6 Sprint 5

6.6.1 Elementos del Backlog

Sprint 5	**PBI_12**: Como usuario quiero crear nuevos archivos en los sistemas de ficheros montados a través del punto de montaje	8 PDH
	PBI_13: Como usuario, quiero recibir un mensaje de error si, al crear un archivo, este ya existe	3 PDH

Tabla 97 Sprint 5

6.6.2 Pruebas realizadas

PBI_12

- **Escenario 1**: En este escenario se prueba la creación de un nuevo documento a través del punto de montaje.

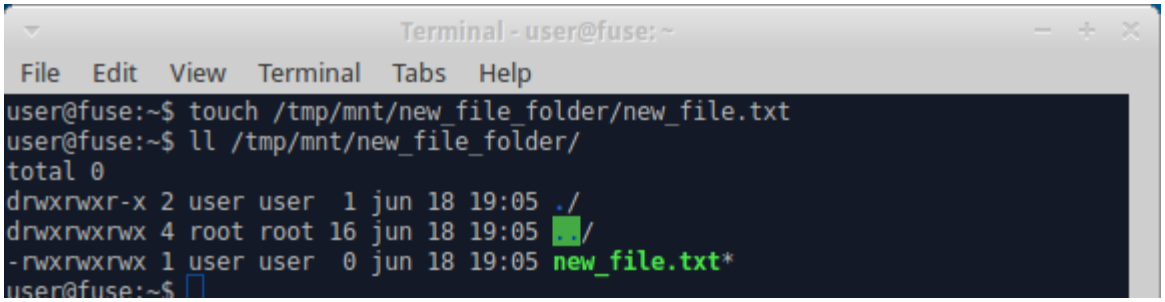


Ilustración 54 Resultado Primer Escenario PBI_12

Ejecutando un comando de creación de ficheros con una ruta relativa al punto de montaje, este es creado en uno de los sistemas de ficheros montados. Los permisos con los que es creado, son de creación, ejecución y lectura para todos los grupos.

- **Escenario 2**: Probamos la asignación de sistemas de ficheros por Round Robin en la creación de ficheros. Para realizar esta prueba, se montan tres sistemas de ficheros en el punto de montaje y se crean cuatro ficheros en la misma carpeta localizada en el punto de montaje.

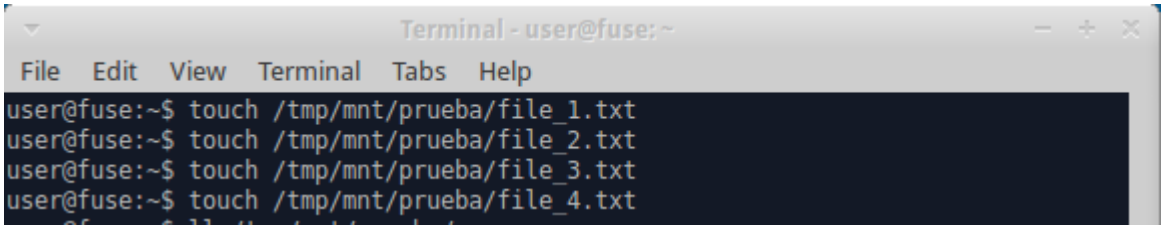


Ilustración 55 Precondición Segundo Escenario PBI_12

Tras ejecutar los comandos, los tres primeros ficheros serán creados en diferentes sistemas de ficheros y el último, en el mismo que el primero.

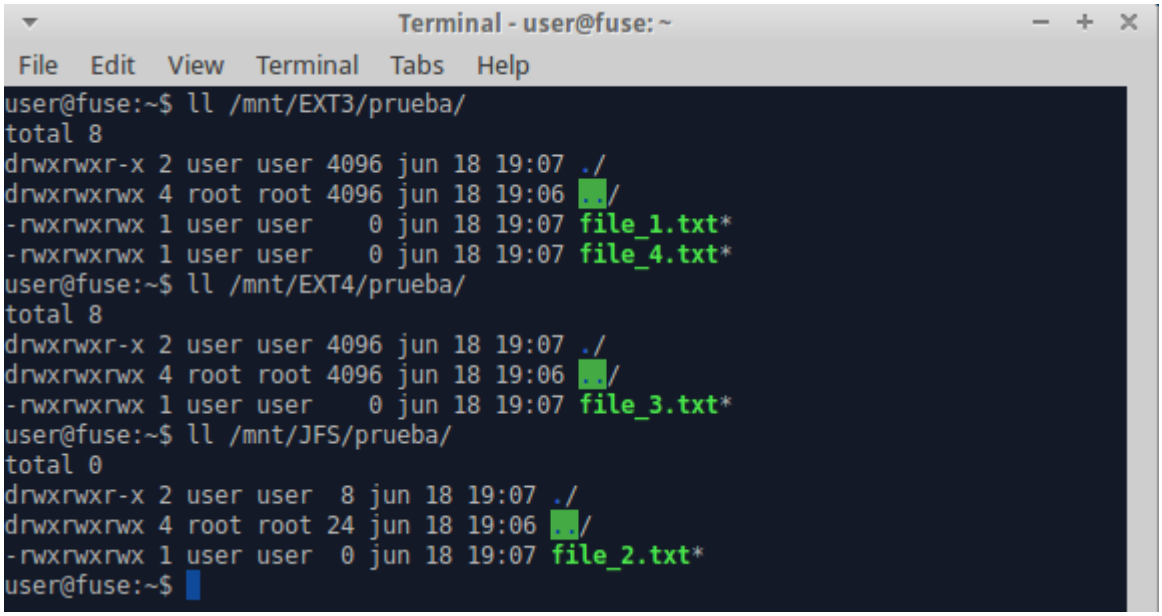


Ilustración 56 Resultado Segundo Escenario PBI_12

PBI_13

- **Escenario 1**: A continuación, se prueba el caso en el que el usuario intenta crear un fichero nuevo en una ruta relativa al punto de montaje que no existe. Ejecutando un comando correspondiente, se obtiene el siguiente error:

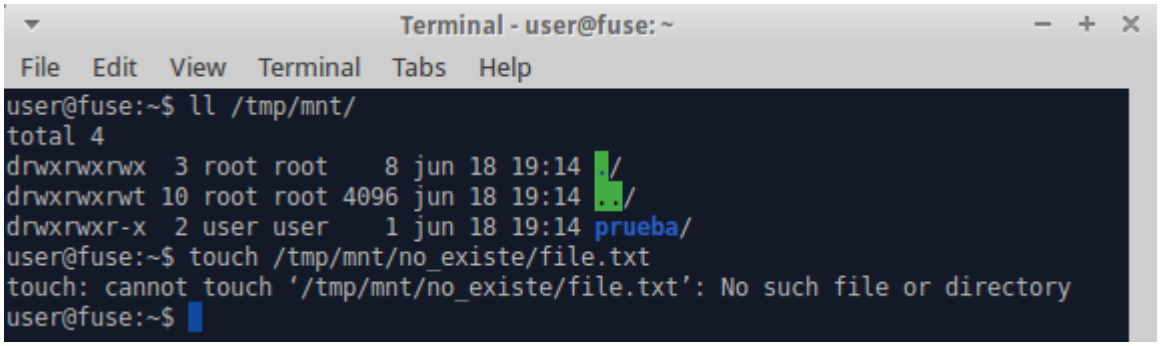


Ilustración 57 Resultado Primer Escenario PBI_13

- **Escenario 2**: En este caso, el usuario intenta crear un fichero con una ruta que ya existe. El sistema localizará el fichero existente y no creará ningún fichero nuevo. Para realizar la prueba, se necesita tener un fichero creado en uno de los sistemas de ficheros:

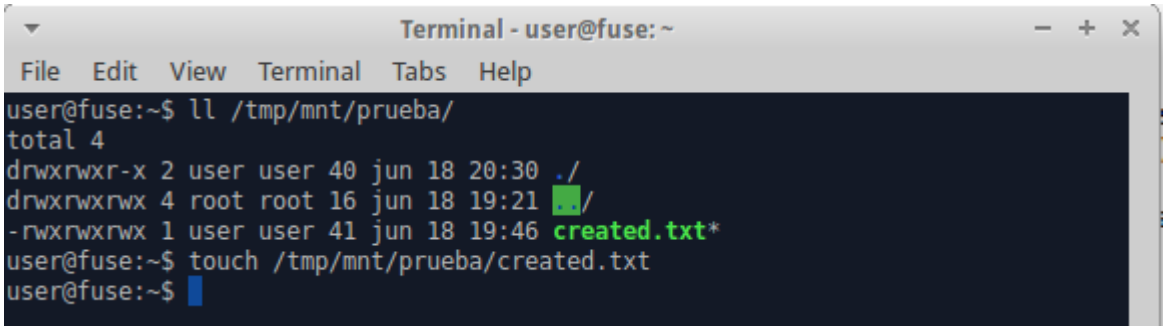


Ilustración 58 Resultado Segundo Escenario PBI_13

Para comprobar que no se ha ejecutado ninguna operación de creación, comprobamos las trazas del fichero debug.txt:

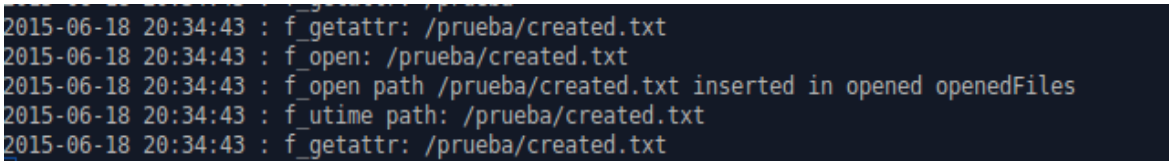


Ilustración 59 Segundo Resultado Del Segundo Escenario PBI_13

6.7 Sprint 6

6.7.1 Elementos del Backlog

Sprint 6	**PBI_14**: Como usuario, quiero abrir archivos alojados en los sistemas de ficheros montados a través del punto de montaje	5 PDH
	PBI_15: Como usuario quiero borrar archivos alojados en los sistemas de ficheros montados a través del punto de montaje	5 PDH

Tabla 98 Sprint 6

6.7.2 Pruebas realizadas

PBI_14

Para ejecutar los escenarios implementados en este PBI, se montarán tres sistemas de ficheros cuyo contenido será el siguiente:

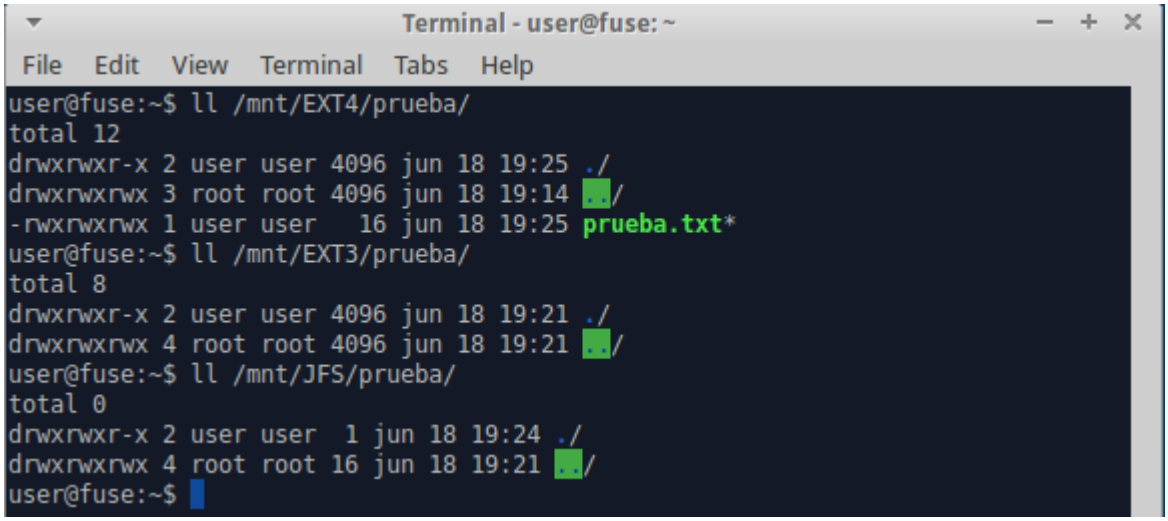


Ilustración 60 Precondición PBI_14

- **Escenario 1**: En este escenario se comprueba que el sistema es capaz de localizar un archivo dentro de los sistemas de ficheros montados en el punto de montaje.

  Cuando se ejecuta un comando que muestra el contenido del fichero de texto "prueba.txt", el sistema lo localiza y devuelve el contenido.

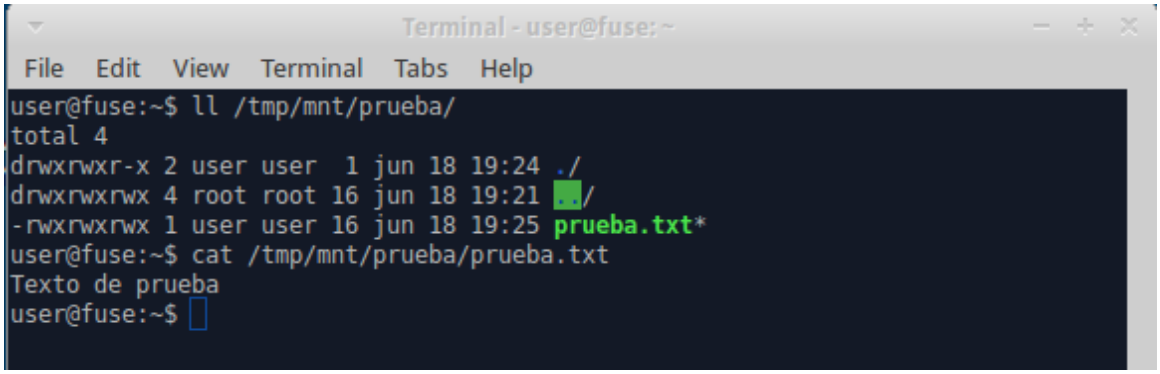


Ilustración 61 Resultado Primer Escenario PBI_14

- **Escenario 2**: A continuación, se muestra el caso en el que se intenta listar el contenido de un fichero que no existe en ninguno de los sistemas de ficheros montados.

  Para esta prueba, se ejecuta un comando que muestre el contenido de un fichero sobre una ruta relativa al punto de montaje que no exista. Como resultado, se obtiene el siguiente error:

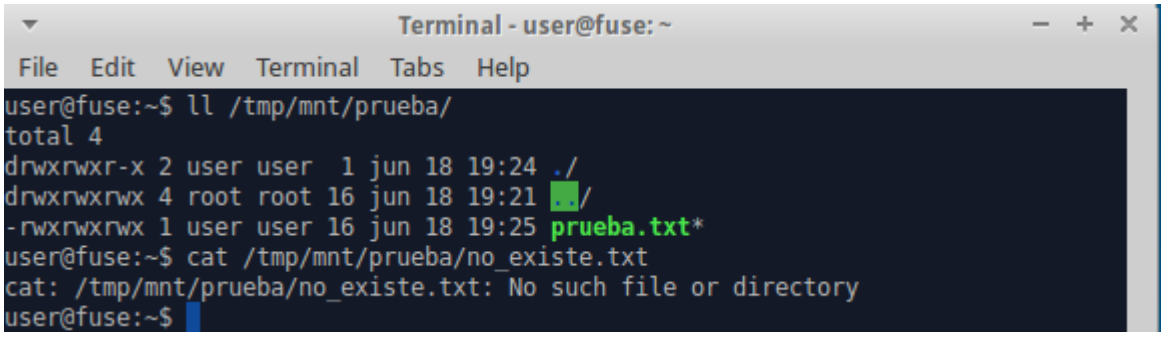


Ilustración 62 Resultado Segundo Escenario PBI_14

PBI_15

Es necesario, para realizar las pruebas descritas en los escenarios del *PBI*, montar tres sistemas de ficheros, con el siguiente contenido:

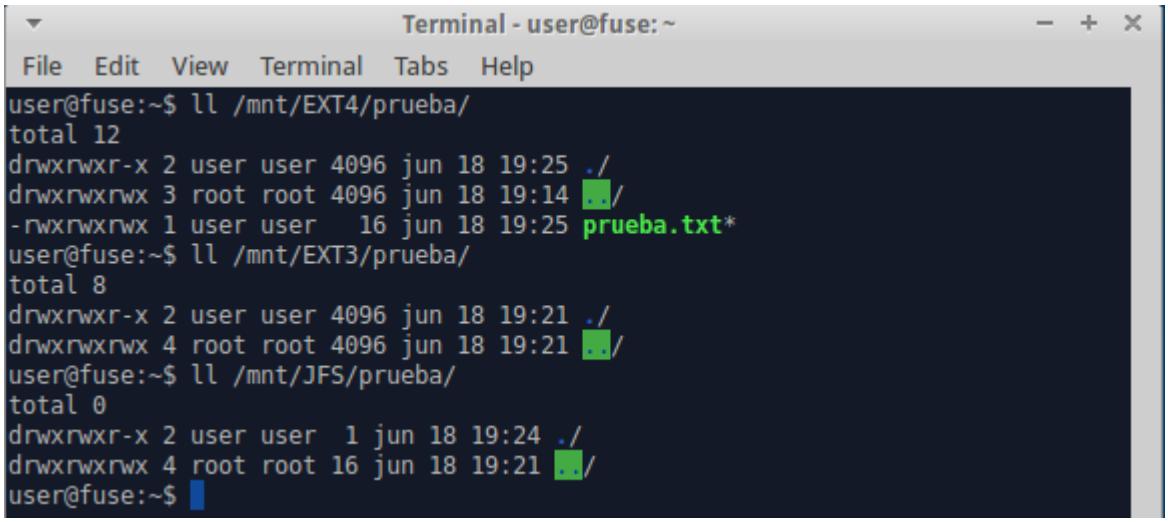


Ilustración 63 Precondición PBI_15

- **Escenario 1**: Este escenario se prueba la capacidad del sistema para localizar y borrar un archivo alojado en uno de los sistemas de ficheros montados.

  Para ello, ejecutamos un comando de borrado de ficheros con la ruta del fichero “prueba.txt”. Al ejecutarlo, el sistema lo localiza y borra sin producir errores

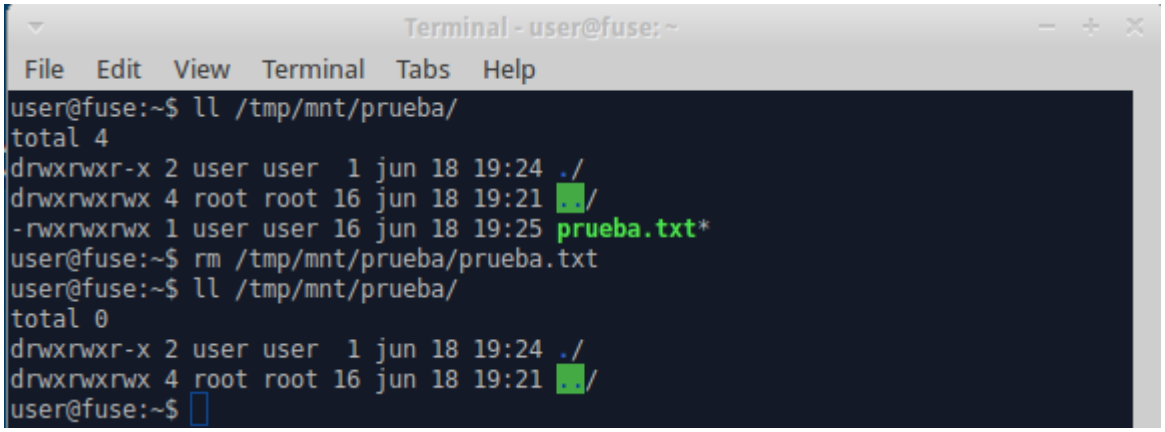


Ilustración 64 Resultado Primer Escenario PBI_15

- **Escenario 2**: En este caso, se prueba la reacción del sistema al no encontrar el fichero que se pretende borrar.

  Ejecutando el comando de borrado con una ruta de fichero que no existe, se obtiene el siguiente error:

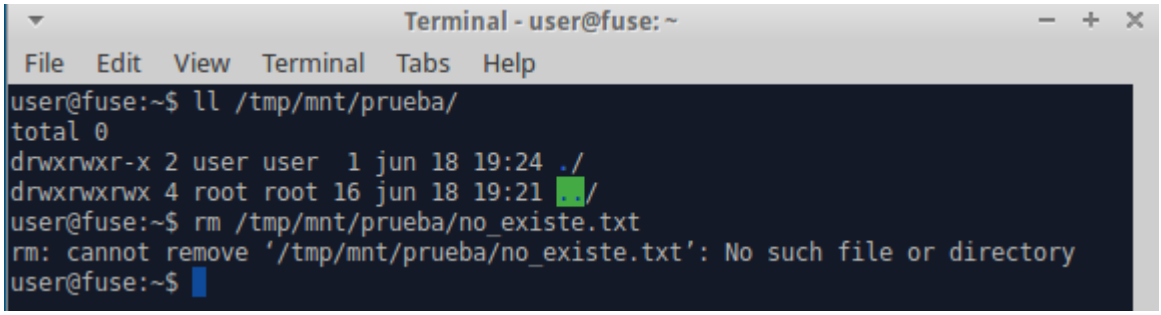


Ilustración 65 Resultado Segundo Escenario PBI_15

6.8 Sprint 7

6.8.1 Elementos del Backlog

Sprint 7	**PBI_16**: Como usuario quiero renombrar archivos y carpetas alojados en los sistemas de ficheros montados a través del punto de montaje.	8 PDH

Tabla 99 Sprint 7

6.8.2 Pruebas realizadas

PBI_16

Para las pruebas del renombrado, se van a montar tres sistemas de ficheros con el siguiente contenido:

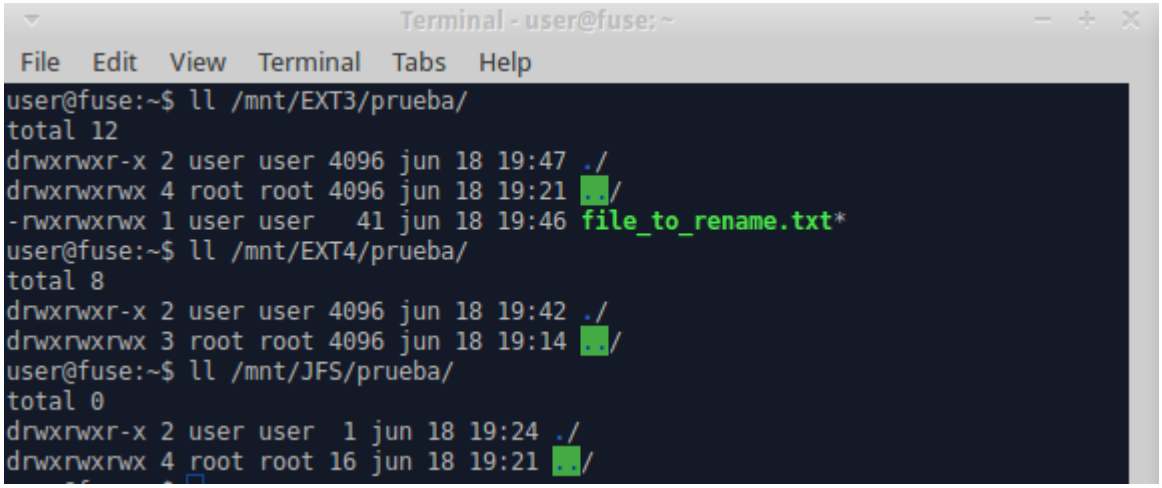


Ilustración 66 Precondición PBI_16

- **Escenario 1**: En este escenario, se prueba la capacidad de renombrado de ficheros del sistema. Para ello, ejecutamos un comando que renombre el fichero “file_to_rename.txt”.

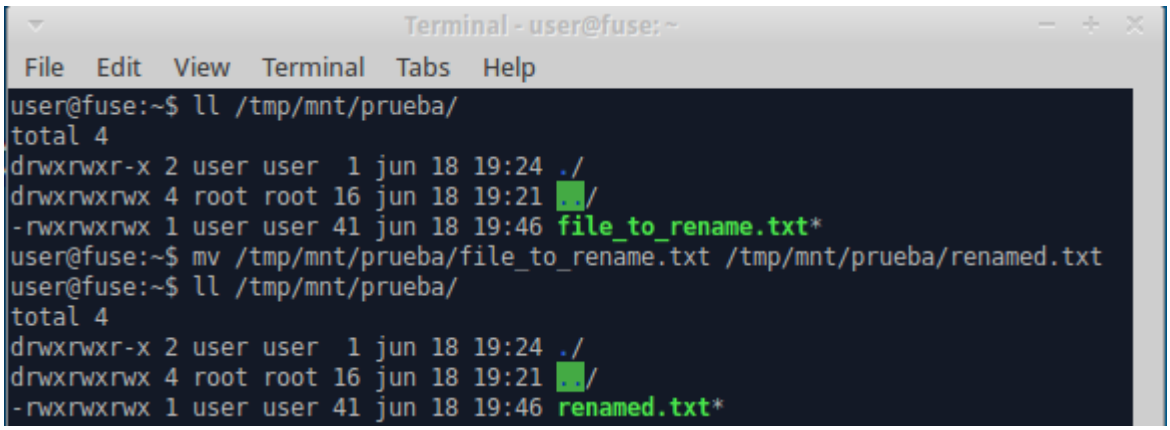


Ilustración 67 Resultado Primer Escenario PBI_16

Tras la ejecución del comando, el fichero ha sido renombrado y permanece en el mismo sistema de ficheros, sin generar ningún duplicado.

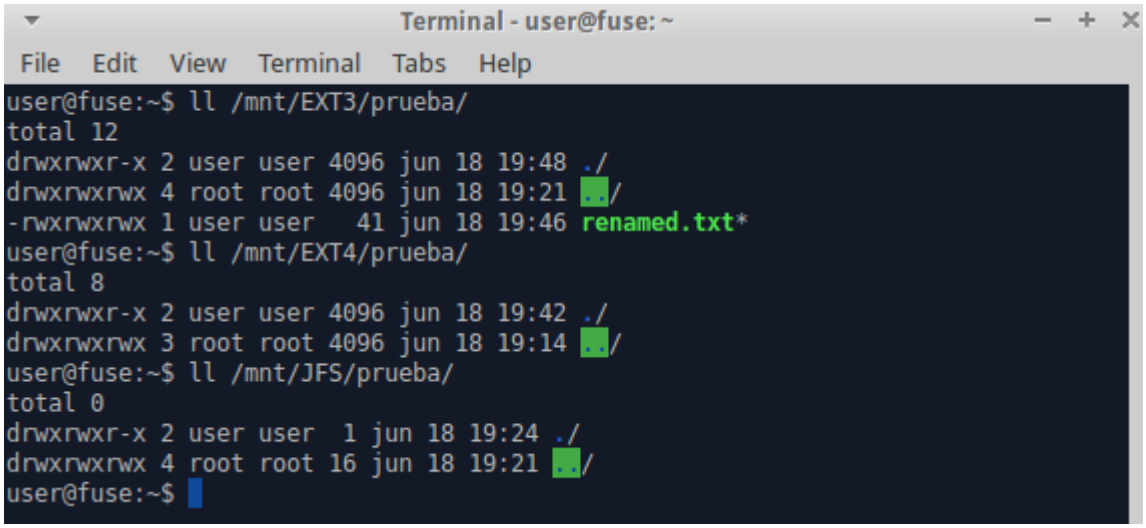


Ilustración 68 Segundo Resultado Primer Escenario PBI_16

- **Escenario 2**: En este caso, se prueba el caso en el que se intenta asignar un nombre que ya existe en la misma ruta a un fichero. Para poder ejecutar este escenario, es necesario mantener un segundo fichero dentro del sistema:

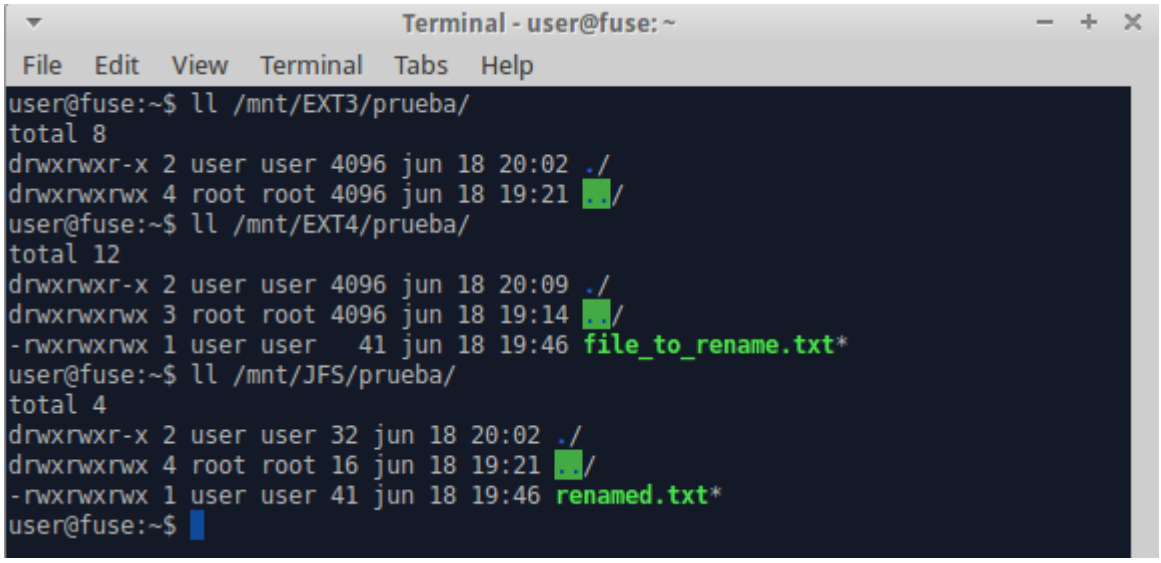


Ilustración 69 Precondición Segundo Escenario PBI_16

Al ejecutar el comando de renombrado, el sistema localiza un fichero con la misma ruta que se desea utilizar para renombrar el fichero “file_to_rename.txt” y muestra el siguiente error:

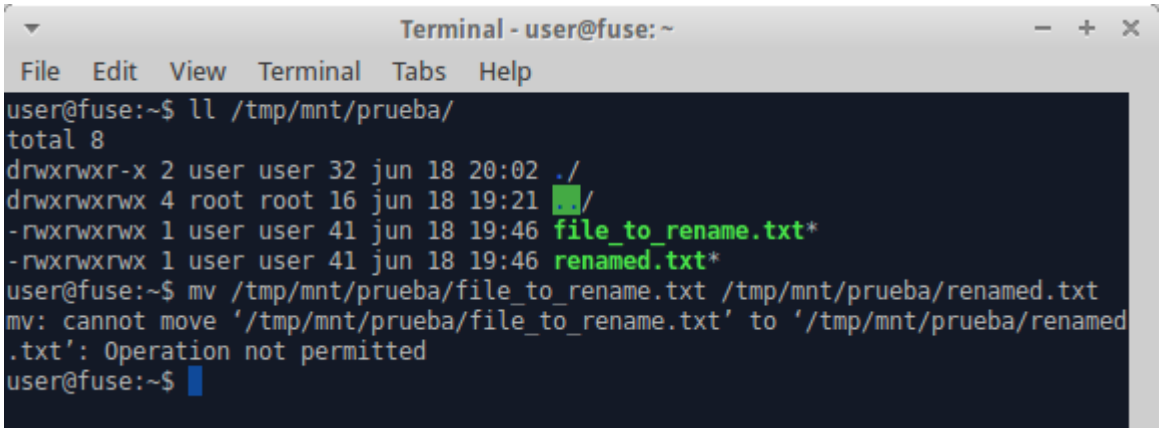


Ilustración 70 Resultado Segundo Escenario PBI_16

- **Escenario 3**: En este escenario se testea la reacción del sistema al intentar renombrar un fichero que no existe. Para ello, ejecutamos un comando de renombrado sobre una ruta relativa al punto de montaje que no exista. El sistema nos devolverá el siguiente error:

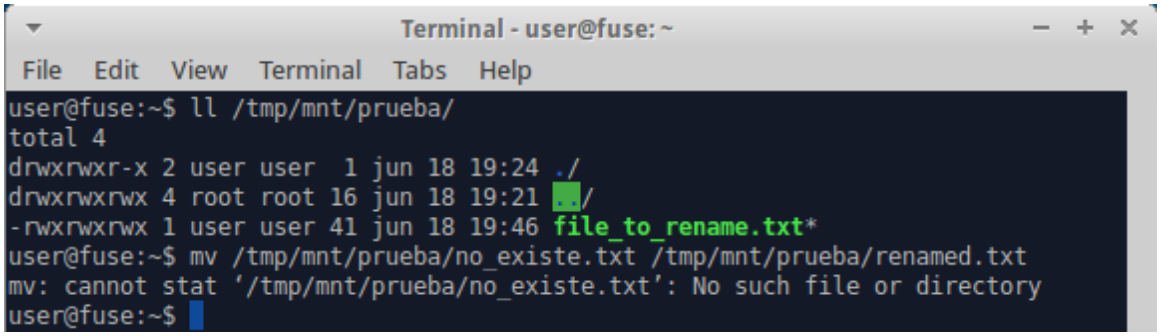


Ilustración 71 Resultado Tercer Escenario PBI_16

7 Presupuesto

En esta sección se mostrará una estimación del coste derivado del desarrollo del proyecto. El siguiente presupuesto está dividido en diferentes categorías en función de la clasificación de los gastos que serán desglosados.

Todos los precios indicados, no incluyen I.V.A.

7.1 Costes de personal

El desarrollo del sistema será realizado a lo largo de siete *sprints* de dos semanas por un grupo formado por dos desarrolladores y un ingeniero de pruebas. El grupo de desarrolladores estará compuesto por un programador junior y un analista programador.

Teniendo en cuenta que el porcentaje de ocupación útil del equipo de desarrollo es de un 70%, el del ingeniero de pruebas es de un 50% y el precio por hora de cada uno de los miembros del equipo, el cálculo de costes de personal será el siguiente:

Categoría	Coste/hora	Horas	Horas Útiles	Total
Analista programador	40,50 €	560	392 (70%)	15.876 €
Programador Junior	30,50 €	560	392 (70%)	11.956 €
Ingeniero de pruebas	38,50 €	560	280 (50%)	10.780 €
Total Personal				38.612 €

Tabla 100 Costes de personal

7.2 Costes de hardware

En esta sección se enumeran los costes de los elementos de hardware necesarios para desarrollar el proyecto.

A estos equipos, se les aplica un periodo de depreciación de 36 meses y un uso dedicado al proyecto del 100% del tiempo.

Concepto	Precio / Unidad	Cantidad	Dedicación	Coste imputable
Dell XPS 13 Developer Edition	949,00 €	3	4 Meses	316,33 €
Total				316,33 €

Tabla 101 Costes de hardware

7.3 Costes de software

Para la realización de este proyecto, se van a utilizar recursos de software licenciados bajo *Open Source Licenses* y gratuitas.

7.4 Presupuesto final

A continuación, se presenta el presupuesto final para el desarrollo del proyecto.

Concepto	Total
Costes de personal	38.612,00 €
Costes de software	0 €
Costes de hardware	316,33 €
Subtotal	38.928,33 €
Riesgo (10 %)	3.892,83 €
Total sin I.V.A.	42.821,16 €
Total (21 % I.V.A. Incluido)	51.813,61 €

Tabla 102 Presupuesto final

8 Conclusiones, mejoras y trabajos futuros

8.1 Conclusiones

Considerando los objetivos del proyecto, establecidos en el capítulo 1.1, se pueden establecer las siguientes conclusiones:

- Se ha alcanzado el objetivo de implementar un núcleo de operaciones sobre directorios. Durante el desarrollo se ha conseguido implementar las operaciones de creación, borrado y acceso.
- Se ha logrado implementar un núcleo de operaciones básicas sobre archivos, que incluye la creación, borrado, renombrado y copiado al sistema de ficheros creado.
- Se ha conseguido crear un módulo para la asignación de sistemas de ficheros con un nivel de acoplamiento con el resto del código muy bajo. Se ha proporcionado una interfaz que puede ser re-implementada para establecer nuevas políticas de asignación siendo necesario, solamente, conocer las rutas de los sistemas de ficheros montados.
- Se ha implementado un sistema de trazabilidad de los eventos ocurridos en el sistema de ficheros así como los errores y operaciones no permitidas.

Además de la consecución de los objetivos secundarios, se ha desarrollado satisfactoriamente, como se demuestra en el capítulo, la primera fase de un sistema que permita al usuario interactuar con un grupo de unidades de almacenamiento montadas previamente.

Es importante mencionar que, debido a la cantidad de documentación encontrada con respecto a las llamadas del sistema y al rendimiento del sistema, C ha sido un lenguaje muy adecuado ya que se ha desarrollado una aplicación mucho más ligera y con mejor tiempo de respuesta de la que se podía haber conseguido utilizando, por ejemplo, JAVA.

En cuanto a los bloqueos encontrados durante el proceso de desarrollo, cabe destacar la calidad de la documentación de la librería *FUSE*. La mayor parte de la documentación que se puede encontrar en internet, describe su comportamiento de una forma superficial, sin centrarse en el detalle de las estructuras que utiliza. Esto ha sido un inconveniente en las primeras iteraciones del desarrollo que se ha ido atenuando en las siguientes.

8.2 Líneas futuras de trabajo

En esta sección se enumeran diferentes propuestas que pueden incrementar el valor del producto de cara a los *Stake Holders*. De esta forma, se aportan nuevas ideas de negocio para su implementación en desarrollos futuros.

- **Gestión del espacio libre en las unidades montadas**: Una de las posibles mejoras a tener en cuenta, es la gestión del espacio libre restante en cada una de las unidades montadas. Esto supondría un cambio en la política de asignación de sistemas de ficheros en la creación y copiado de ficheros.

  Además del control, a nivel de aplicación del espacio restante en las unidades, sería necesario la implementación de un algoritmo que, en el caso de no restar espacio suficiente en el sistema de ficheros designado por defecto, asignase un punto diferente de guardado en función del espacio restante o, devolviese un error.

- **Asignación de diferentes políticas de almacenamiento mediante parametrización**: Una mejora interesante supondría la posibilidad de que la aplicación permitiese diferentes políticas de almacenaje de los ficheros en función de un parámetro en la aplicación. De esta forma, podrían seleccionarse políticas diferentes a *Round Robin* a la hora de montar los sistemas de ficheros en el punto de montaje.

- **Implementación de las operaciones necesarias para mover ficheros dentro del punto de montaje**: Actualmente, el sistema no permite copiar ficheros de una ruta relativa al punto de montaje a otra. Una posible línea de trabajo puede basarse en implementar las operaciones y establecer las políticas necesarias para poder realizar esta acción.

- **Implementación de las operaciones necesarias para permitir la gestión de permisos y usuarios**: Con la implementación actual, los ficheros y carpetas se crean con permisos de escritura, lectura y ejecución para todos los usuarios y grupos. Sería interesante, de cara a la securización del sistema, implementar las operaciones correspondientes a la asignación de permisos y propietarios.

- **Adaptación del sistema para montar diversos sistemas de sincronización de datos con servicios en la nube**: Una aplicación interesante del sistema es la posibilidad de integrar diferentes sistemas de almacenamiento en la nube en una sola carpeta. De esta forma, se podría repartir el contenido entre diferentes servicios como *Google Drive*,

Dropbox o *Box*. Supondría, la implementación de un sistema de autenticación unificado para los diferentes sistemas y la gestión de los ficheros temporales que, los clientes de estos servicios, crean para la sincronización de datos.

9 Acrónimos y Abreviaturas

- API : Interfaz de programación de aplicaciones. Se trata de un mecanismo para conseguir abstracción en la programación. Consiste en exponer una serie de funciones de uso general que serán accesibles por los colaboradores del componente.
- CFM: *Common file model* o modélo común de archivos.
- EXT2: *Second extended filesystem* o segundo sistema de archivos extendido.
- EXT3: *third extended filesystem* o tercer sistema de archivos extendido.
- EXT4: *fourth extended filesystem* o cuarto sistema de archivos extendido.
- FUSE: *File System in User Space* o sistema de ficheros en espacio de usuario.
- GNU: Acrónimo recursivo de *GNU* no es Unix.
- GPL: Licencia Pública General de *GNU*.
- JFS: *Journaling File System*. Es un sistema de ficheros respaldado por IBM.
- PBI: *Product Backlog Item* o item de la pila de producto.
- VFS*: Virtual Filesystem* o sistema de ficheros virtual.
- XFS: Sistema de ficheros creado por *SGI*.